МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНЕГО
СПЕЦИАЛЬНОГО ОБРАЗОВАНИЯ УССР

ОДЕССКИЙ ОРДЕНА ТРУДОВОГО КРАСНОГО ЗНАМЕНИ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. И. И. МЕЧНИКОВА

В. С. МАКОГОН

ПРАКТИКА ПРОГРАММИРОВАНИЯ НА БЕЙСИКе

Учебное пособие

Одесса ОГУ 1987

МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНЕГО СПЕЦИАЛЬНОГО ОБРАЗОВАНИЯ УССР
ОДЕССКИЙ ОРДЕНА ТРУДОВОГО КРАСНОГО ЗНАМЕНИ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. И.И.МЕЧНИКОВА

В.С.МАКОГОН

ПРАКТИКА ПРОГРАММИРОВАНИЯ НА БЕЙСИКе

Учебное пособие

Утверждено ученым советом института

Одесса ОГУ 1987

УДК 519.95

Практика программирования на БЕЙСИКе: Учеб. пособие /В.С.Макогон. - Одесса: ОГУ, 1987. - 95 с.

В пособии излагаются основные принципы программирования на БЕЙСИКе; приводятся примеры, на которых построено обучение программированию.

Предназначено для начинающих пользователей - студентов механико-математических и других естественных факультетов по необеспеченному учебной литературой разделу курса "Вычислительные машины, мини- и микрокомпьютеры".

Ил. 47. Табл. I. Библиогр.: 8 назв.

Ответственный редактор Н.Я.Тихоненко

Рецензенты В.Н.Гребенкин, В.В.Эфендиев,
кандидаты физико-математических наук

© Одесский ордена Трудового Красного Знамени
государственный университет им. И.И.Мечникова

ВВЕДЕНИЕ

Кардинальное ускорение научно-технического прогресса в нашей стране немыслимо без широкого внедрения средств информатики и вычислительной техники во все сферы человеческой деятельности.

В последние годы сформировались новые области массового применения компьютерной техники. В частности, современные мини- и микрокомпьютеры все шире внедряются в систему высшего и среднего образования, причем они становятся не только и не столько средством обучения программированию, сколько универсальным инструментом профессиональной подготовки специалистов. Эффективное обучение современного специалиста компьютерной грамотности - одна из самых актуальных проблем современности.

Успешное овладение программированием как ремеслом немыслимо без непосредственного общения человека с компьютером. Применение в учебном процессе больших (типа ЕС ЭВМ) вычислительных машин, в которых традиционно используется пакетный режим обработки заданий, лишает пользователя непосредственного доступа к машине и делает невозможным оперативное внесение в программу изменений по результатам ее отладки. А наличие такой возможности особенно полезно для начинающих программистов, так как длительное ожидание результатов решения задачи на компьютере, характерное для пакетного режима, отрицательно сказывается на эффективности процесса обучения.

Для современных мини- и микрокомпьютеров характерна обработка программ в диалоговом режиме: процесс редактирования программы, ее отладка и выполнение проходят в режиме активного взаимодействия человека с машиной.

Одним из самых популярных диалоговых языков программирования является БЕЙСИК. Он предназначен для решения математических и инженерно-технических задач и входит в состав программного обеспечения персональных компьютеров, применяемых в учебном процессе.

В данном пособии рассмотрено подмножество языка, ориентированное на реализацию несложных алгоритмов, соответствующих школьному курсу информатики. Все приведенные программы, не считая простейших, проверены и отлажены на персональном компьютере *ЯМАХА MSX* и могут без всяких изменений быть использованы (ввиду выбранного подмножества языка) на отечественных ЭВМ, в частности, на микроЭВМ "Искра-226", ДВК, "Электроника ДЗ-28" и др.

I. ОРГАНИЗАЦИЯ БЕЙСИК-ПРОГРАММ. ОСНОВНЫЕ КОНСТРУКЦИИ ЯЗЫКА

Организация и выполнение БЕЙСИК-программ на ЭВМ осуществляются под управлением служебной программы - интерпретатора. Программа представляет собой запись алгоритма решения задачи в виде, понятном для интерпретатора БЕЙСИКа. Программа и данные вводятся в машину обычно с клавиатуры терминала (дисплея), интерпретируются и результаты снова выдаются на экран дисплея либо на печать. При включенном терминале на его экране постоянно высвечивается специальный символ - курсор, указывающий на ту позицию экрана, в которую будет занесен очередной символ с клавиатуры или из ЭВМ. После ввода (вывода) очередного символа курсор автоматически смещается на одну позицию вправо. В состав клавиатуры входят клавиши для набора текста и служебные клавиши. При нажатии любой из клавиш для набора текста изображенный на ней символ заносится в ту позицию экрана, на которую указывает курсор. На каждой из этих клавиш изображен один или несколько символов. (Чаще всего их два: один в верхней, а другой - в нижней части клавиши.) Нужный символ выбирают, предварительно нажав клавишу выбора регистра (*РУС*, *SHIFT* , *CAPS*). При коротком нажатии одной из клавиш происходит однократный вывод символа. Длительное (свыше двух секунд) нажатие приводит к многократному выводу соответствующего символа, пока не будет отпущена нажатая клавиша. Переход на новую строку и возврат курсора в начало строки осуществляется специальной клавишей "Ввод"*.

* В зависимости от типа компьютера эта клавиша обозначается по-разному: ⏎ , *CR/LF* , *ПС* и т.п.

В основе языка программирования БЕЙСИК лежит понятие оператора. Под оператором понимается конструкция языка, которая определяет характер выполняемого действия либо описывает свойства объектов программы. Операторы включают в себя как служебные слова, которые имеют строго определенное начертание и смысл, так и некоторые параметры пользователя (тело оператора). По мере необходимости будем вводить служебные слова (они пишутся на английском языке) и пояснять их смысл и назначение.

Интерпретатор БЕЙСИКа может использоваться в режиме прямого выполнения команд и в режиме выполнения программы (*RUN*). В первом случае перед оператором номер строки не ставится и выполняется он сразу после ввода. Результаты арифметических или логических операций могут быть запомнены и (или) выведены на экран; сами команды после исполнения нигде не запоминаются. Прямой режим полезен при отладке программ и при использовании БЕЙСИК-интерпретатора в качестве калькулятора для вычислений, не требующих сложных программ.

Режим *RUN* используется для подготовки, запоминания и последующего исполнения программ.

I.I. Подготовка БЕЙСИК-программы

По своей структуре БЕЙСИК-программа представляет последовательность пронумерованных строк. В качестве разделителя строк используется символ "Ввод" ("возврат каретки"/"перевод строки").

Каждая строка программы начинается с ее номера, который представляет целое число без знака, содержащее обычно от одной до четырех-пяти цифр. Номер строки применяется для ее идентификации и может быть использован для ссылки на эту строку; кроме того, он служит для упорядочения строк программы при ее вводе на ЭВМ. Последнее означает, что строки можно вводить в любом порядке. Перед началом выполнения программы машина сортирует строки по возрастанию номеров. Это облегчает изменение и исправление программы. Номера строк выбираются произвольно, но с некоторым шагом (равным, например, 5 или I0) для того, чтобы позже можно было вставить дополнительные строки между уже имеющимися, не нарушая общей упорядоченности номеров.

Максимальная длина строки зависит от типа ЭВМ. Однако при вводе программы через клавиатуру дисплея рекомендуется ограничивать ее длину видимой строкой экрана дисплея. Строка программы может состоять

из одного или нескольких операторов; в последнем случае они отделяются друг от друга символом ":" (двоеточие). Пробел ни до двоеточия, ни после него не ставится.

В произвольных местах строки, в пределах ее допустимой длины могут быть записаны пробелы, которые используются для удобочитаемости программ. Все служебные слова, в том числе и такие, как *GOTO*, *GOSUB*, записываются без пробелов.

При подготовке программы на экране дисплея строка считается введенной в память ЭВМ после нажатия клавиши "Ввод". Изменить что-либо в такой строке можно, заново набрав эту строку с тем же номером. Аналогично добавляются новые строки в программу. Если же ошибка обнаружена до нажатия клавиши "Ввод", можно вернуться к месту ошибки, нажимая последовательно клавишу *BACK SPACE (BS)*, исправить ошибку и продолжить дальше набор содержимого подготавливаемой строки. В современных микроЭВМ и персональных компьютерах используются и более совершенные средства редактирования текста на экране. Чтобы убрать вообще некоторую строку из программы, достаточно набрать ее номер и нажать "Ввод", что равносильно вводу пустой строки.

Для управления положением курсора на экране используются клавиши → и ← (для смещения курсора вправо или влево) и клавиши ↑ и ↓ (для смещения курсора по вертикали вверх или вниз).

Чтобы удалить всю хранящуюся в памяти программу, достаточно ввести директиву *CLEAR* (*NEW* в ПК *YAMAHA*). С нее обычно начинается подготовка новой БЕЙСИК-программы.

После того как вся программа введена в память машины и отредактирована, можно приступать к ее выполнению. Пуск программы со строки с наименьшим номером осуществляется с помощью команды *RUN*, для чего последовательно нажимаются клавиши *RUN* и "Ввод". В общем случае программа может быть запущена с любой строки, т.е. команда *RUN* может использоваться в виде

RUN <номер строки>

где <номер строки> указывает строку, с которой происходит запуск программы.

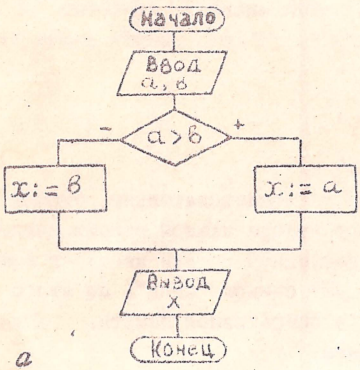
Подготовленную в памяти БЕЙСИК-программу в любой момент можно вызвать на экран дисплея, набрав директиву *LIST* и нажав клавишу "Ввод".

Выполнение программы осуществляется в порядке возрастания номеров строк, если не предусмотрено программное изменение этой последовательности.

I.2. Пример программы

Знакомство с языком программирования БЕЙСИК начнем с рассмотрения программы для поиска большего из двух чисел.

Блок-схема алгоритма изображена на рис. I.I,а, соответствующая БЕЙСИК-программа - на рис. I.I,б.



а

```
1Ø REM МАКСИМ (A,B)
2Ø INPUT A,B
3Ø IF A>B THEN 5Ø
4Ø LET X=B:GOTO 6Ø
5Ø LET X=A
6Ø PRINT "MAX=";X
7Ø END
```

б

Рис. I.I

Прежде всего замечаем, что каждый оператор начинается со служебного слова, которое следует после номера строки. Это слово (например, *INPUT*, *IF*, *LET*, *PRINT*, *END*) указывает тип и назначение оператора. Рассмотрим более подробно приведенную на рис. I.I,б программу.

Первый оператор (строка *1Ø*) - комментарий; он присваивает название программе и может в общем случае отсутствовать. Второй оператор (строка *2Ø*) *INPUT A, B* - оператор ввода исходных данных, который в процессе выполнения программы обеспечит присвоение переменным *A*, *B* конкретных числовых значений, вводимых с клавиатуры дисплея.

В строке *3Ø* проверяется условие *A > B*. Если оно истинно, то машина должна перейти на строку *5Ø*, где переменной *X* присваивается введенное ранее числовое значение *A*; в противном случае машина переходит к следующему оператору (строка *4Ø*). Таким образом, условный оператор *IF-THEN* сообщает машине, куда следует перейти в зависимости от истинности или ложности некоторого условия.

Строка 4Ø содержит два оператора: первый присваивает переменной X числовое значение переменной B ; второй (GOTO 6Ø) передает управление на строку 6Ø , обходя таким образом оператор LET X=A , выполняющийся только тогда, когда A>B . Содержащийся в этой строке оператор PRINT вызовет печать результата. Тело оператора организовано так, что в строку экрана дисплея (или на печать) будет выдан поясняющий текст MAX= , а затем числовое значение X - найденный результат. Так, если оператору INPUT ввести значения 7 и 3, то ответ будет выдан в виде

MAX= 7

Оператор END завершает программу.

Программа вводится на выполнение последовательно, строка за строкой с клавиатуры дисплея: после набора каждой строки следует нажать клавишу "Ввод". После ввода последней строки вводится команда RUN (пуск), за которой также следует символ "Ввод". До этого момента машина запоминала программу в оперативной памяти. RUN - команда машине на выполнение программы.

I.3. Алфавит языка

В алфавит БЕЙСИКа входят следующие символы:

а) заглавные латинские буквы - A,B,C,... X,Y,Z ;

б) десятичные цифры - 0, I, 2,...,9;

в) знаки препинания

. (точка)
; (точка с запятой)
: (двоеточие)
, (запятая)
' (апостроф)
" (кавычки);

г) знаки арифметических операций

+ (сложение)
- (вычитание)
* (умножение)
/ (деление)
↑ или ^ (возведение в степень);

д) знаки операций отношения

= (равно)
< (меньше)
<= (меньше или равно)
> (больше)
>= (больше или равно)
<> (не равно);

е) специальные символы

( ) = ␣ (пробел*).

При записи комментариев (оператор *REM*) и в литерных константах (любой текст, заключенный в кавычки) можно использовать заглавные русские буквы, например:

```
10 REM ПОИСК МАКСИМ.ЧИСЛА
20 PRINT"МАКСИМАЛЬНАЯ ВЕЛИЧИНА=",X
```

I.4. Числа

В рассматриваемом подмножестве языка БЕЙСИК определен лишь вещественный (действительный) тип чисел. При этом числа могут быть представлены в программе в форме целого числа, в форме с фиксированной точкой и в форме с плавающей точкой. Знак "+" перед числом не ставится, а для отделения целой части числа от дробной при записи десятичных дробей используется точка. Запись целых констант по существу не отличается от общепринятой математической записи. Нулевую целую часть числа с фиксированной точкой можно опускать. При записи констант цифра Ø перечеркивается.

Примеры записи чисел:

целые	с фиксированной точкой
Ø	.Ø7
I95	-I9.357
␣I3	24.

При представлении чисел в форме с плавающей точкой последовательно записываются знак числа (для отрицательных чисел), мантисса,

* Символ "пробел", изображаемый на бумаге как ␣ , на клавиатуре никак не обозначается.

латинская буква *E*, знак порядка и порядок. Например, одно и то же число 0,007382 может быть представлено несколькими различными способами:

Ø.7382 E - 2
Ø.ØØØØ 7382 E 2
73.82 E -4

Здесь буква *E* имеет смысл "возвести 10 в степень". Показатель степени (порядок) может не иметь знака, если он положителен. Но нельзя задавать *E4* в качестве числа 10000; следует писать *1 E 4* для того, чтобы указать, что единица умножается на 10^4.

Разрядность чисел в микроЭВМ "Искра-226" не превосходит 13 десятичных разрядов. Диапазон целых чисел - от 0 до 7999, дробных - от 10^{-99} до $(1-10^{-12})\cdot 10^{99}$. Диапазон значений целых чисел ПК "Ямаха" от -32768 до 32767, вещественных - от 10^{-64} до 10^{63}.

1.5. Переменные и массивы

Величины, значения которых в процессе выполнения программы могут изменяться, называются переменными. Числовая переменная в БЕЙСИКе обозначается любой буквой латинского алфавита, или любой латинской буквой, за которой следует одна цифра. Например:

A, N5, B1

Совокупность букв и цифр, обозначающих переменную, называют ее именем (идентификатором). Переменная в программе означает величину, значение которой программист не знает во время написания программы. Операторами *LET*, *READ*, *INPUT* переменным даются (или присваиваются) конкретные значения.

Присвоенные таким образом значения переменных не изменяются до тех пор, пока снова не встретятся операторы *LET*, *READ*, *INPUT* с новыми значениями для этой переменной. Однако все переменные считаются неопределенными до начала выполнения программы. Переменной необходимо всегда присваивать значения перед тем, как использовать ее в дальнейших вычислениях.

Кроме рассмотренных переменных, которые носят название простых (или одиночных) переменных, в БЕЙСИКе широко применяются переменные с индексами, служащие для обозначения элементов массивов (под массивом понимают упорядоченную совокупность однородных величин, которой присвоено одно общее имя). Имена массивам присваиваются так же, как и простым переменным.

Переменная с индексом обозначается именем массива, за которым в круглых скобках указываются разделенные запятыми индексы - числовые или буквенные. Например:

$$A(3),\ B1(1,3),\ Д(K,1),\ X(1)$$

Положение каждого элемента в массиве определяется с помощью индексов. Количество индексов, однозначно определяющих элемент массива, задает его размерность. В БЕЙСИКе допускается использование только одномерных и двумерных массивов. Для одномерного массива индекс определяет порядковый номер элемента в массиве; для двумерного массива первый индекс - номер строки, второй - номер столбца, на пересечении которых находится элемент. Элементы одномерного массива, а также строки и столбцы двумерного нумеруются с нуля.

Примеры записи элементов массивов:

Математическая запись	Запись на БЕЙСИКе
$a_0, a_1, a_2, a_3 \ldots a_m$	$A(Ø), A(1), A(2), A(3) \ldots A(M)$
$b_{00}, b_{01}, b_{02}, b_{03}$	$B(Ø,Ø), B(Ø,1), B(Ø,2), B(Ø,3)$
$b_{10}, b_{11}, b_{12}, b_{13}$	$B(1,Ø), B(1,1), B(1,2), B(1,3)$

В общем случае индексом при задании переменной с индексами может быть произвольное арифметическое выражение. За значение индекса принимается целая часть значения арифметического выражения. Например, $B(1*J,K)$ при значениях переменных $J = 2.9$, $J = 2$, $K = 3.7$ определяет элемент массива B, стоящий на пересечении его 5-й строки и 3-го столбца.

Если в программе используются переменные с индексами, то соответствующие массивы должны быть объявлены с помощью специальных операторов (см. гл. 2) прежде, чем эти переменные будут использованы в программе.

I.6. Стандартные функции языка

При решении задач нередко требуется вычисление математических функций - тригонометрических функций ($\sin x$, $\cos x$, $arctg\, x$ и т.д.), модуля числа, логарифмов, e^x и др. Для их вычисления в БЕЙСИКе предусмотрены стандартные функции, каждая из которых обозначается соответствующим именем, за которым в круглых скобках следует аргумент.

Перечень наиболее употребительных стандартных функций БЕЙСИКа приводится в таблице.

Стандартная функция БЕЙСИКа	Математическая функция	Примечание
ABS (x)	$\|X\|$	
ATN (x)	arc tg x	
COS (x)	cos x	
EXP (x)	e^x	
INT (x)	ent (x)	Наибольшее целое, не превосходящее X
RND (x)		Случайное число между 0 и I
SIN (x)	sin x	
SOR (x)	$\sqrt{x}$	$sign X = \begin{cases} I, & \text{если } X>0 \\ 0, & \text{если } X=0 \\ -I, & \text{если } X<0 \end{cases}$
SGN (x)	sign x	
TAN (x)	tg x	
LOG (x)	ln x	

Аргументом X может быть число, переменная, арифметическое выражение (в том числе содержащее функции). Тригонометрические функции вычисляются для аргументов, выраженных только в радианах; значения обратных тригонометрических функций получаются в радианах.

Выражение вида COS (X) , т.е. имя стандартной функции вместе с заданным значением аргумента, называют указателем функции.

I.7. Арифметические выражения

Константы, переменные и указатели функций, соединенные между собой знаками арифметических операций с использованием, быть может, скобок, образуют арифметическое выражение. Например:

Математическая запись	Запись на БЕЙСИКе
$4a/b-c$	4*A/B-C
ax^2+bx+c	A*X^2+B*X+C
$x_i^2+y_i^2$	X(I)^2+Y(I)^2
$\sqrt{(ax+b)/(cx-d)}$	SQR(A*X+B)/(C*X-D))
$\ln\sqrt{x}e^y$	LOG(SQR(X*EXP(Y)))
$\frac{-b-\sqrt{b^2-4ac}}{2a}$	(-B-SQR(B^2-4*A*C))/(2*A)

Порядок выполнения операций в арифметических выражениях определяется следующими правилами.

I. Выражение внутри скобок вычисляется до того, как взятая в скобки величина будет использована в дальнейших вычислениях.

2. Внутри скобок и в бесскобочных записях порядок действий совпадает с общепринятым: сначала вычисляются значения функций и выполняются все возведения в степень, затем умножения и деления и, наконец, сложения и вычитания. Деление имеет тот же приоритет, что и уменьшение, а вычитание - тот же, что и сложение.

3. При отсутствии скобок в арифметическом выражении операции, имеющие одинаковый приоритет, выполняются в порядке их следования слева направо.

4. Все знаки операций в выражении должны быть записаны явно. Нельзя записывать подряд два знака операций. В этом случае они должны быть разделены круглыми скобками. Например:

$$A/(-B),X\hat{}(-Y)$$

Эти правила иллюстрируются следующими примерами.

Выражение $X+COS(Y)*Z\hat{}T$ вычисляется так: Z возводится в степень T, полученный результат умножается на $COS(Y)$, а затем X складывается с результатом умножения, что не отличается от обычного порядка выполнения арифметических действий. Если это не тот порядок, который требуется, необходимо использовать круглые скобки для указания желаемого порядка вычислений. Например, если необходимо возвести в степень T произведение $COS(Y)*Z$, то нужно записать $X+(COS(Y)*Z)\hat{}T$ или, если необходимо умножить $X+COS(Y)$ на Z в степени T, то нужно записать $(X+COS(Y)*Z\hat{}T$.

Правила также говорят о том, что машина, встретив $A-B+C$, будет вычитать B из A, а затем C складывать с этой разностью; встретив $A/B*C$, она разделит A на B и полученное частное умножит на C; встретив $A\hat{}B\hat{}C$, она возведет число A в степень B, а затем полученный результат возведет в степень C.

Следует помнить такое "золотое" правило: если возникают какие-либо сомнения относительно приоритета выполняемых действий, рекомендуется использовать побольше скобок, чтобы избежать возможной двусмысленности.

Особую категорию выражений в языке БЕЙСИК представляют условия, которые являются логическими выражениями и принимают лишь два значения - "истина" и "ложь". Условие состоит из левой и правой частей, соединенных между собой одним из знаков операций отношения (см. разд. I.3):

$$=,\ <,\ <=,\ >,\ >=,\ <>$$

Левой и правой частью условия может быть произвольное арифметическое выражение. Примеры:

Математическая запись	Запись на БЕЙСИКе
$b^2-4ac\geq 0$	`B*B-4*A*C>=0`
$y_i<m$	`Y(I)<M`

Упражнения

1. Записать на БЕЙСИКе следующие числа:

а) 37,423 б) -98 в) 0,000073
г) 10^{-5} д) $-9{,}73\cdot10^3$ е) 47000000

2. Выделить среди заданных чисел форматы с фиксированной и плавающей точкой:

а) 19.731 б) -0.971 в) .2034E-2
г) 1E-3 д) -.07 е) -3.72453E2

3. Записать на БЕЙСИКе такие арифметические выражения:

а) $\dfrac{2x^3-b_i}{3by}$ б) $(0{,}975+3\operatorname{tg}^2\alpha)^3+2^{\cos\beta}$

в) $\sqrt{\sin^2x+e^x\ln y}$ г) $\dfrac{10^{-5}+0{,}3\sqrt[3]{\ln^2z}}{|x^2+y^2|}$

4. Записать на языке обычных математических выражений:

а/ AXX^2+BXB-C)/D+5XZ

б/ (1+EXP(ABS(A^2-BXB)))^(1/5)

в/ LOG(SIN(ABS(X(1)+Y(I))))

5. Пусть $A = 2$, $B = 3$, $C = 4$. Определить истинность следующих условий:

а/ BXB-4XAXC>0

б/ A+BXB>=C^2

в/ INT(C/B)=B-A

г/ SGN (A-B)=A-B

2. ОСНОВНЫЕ ОПЕРАТОРЫ ЯЗЫКА БЕЙСИК

Программа на языке БЕЙСИК представляет собой последовательность операторов. По своей структуре оператор - это некоторая стандартная запись, предписывающая машине выполнение определенных действий (исполнимый оператор) или содержащая некоторую осведомительную информацию об объектах программы (неисполнимый оператор).

Для обозначения общих понятий будем использовать угловые скобки. Так, <переменная> указывает на имя, состоящее из одной буквы и, быть может, следующей за ней цифры.

2.1. Оператор присваивания

Наиболее употребительный оператор, обеспечивающий в программе вычислительные операции и вызывающий изменение значений переменных, - это оператор присваивания. Здесь и далее под переменной будем понимать как простую переменную, так и переменную с индексами. Оператор присваивания имеет следующий формат:

LET<переменная> = <арифметическое выражение>

Этот оператор не является оператором алгебраического равенства, а является командой для ЭВМ: вычислить значение стоящего справа от знака "=" арифметического выражения при текущих значениях входящих в него переменных и результат присвоить переменной, стоящей слева от знака "=". В частности, арифметическим выражением может быть число или отдельно взятая переменная.

Заслуживает внимания случай, когда справа от знака присваивания в выражении присутствует переменная, стоящая и слева от него. Пусть, например, по ходу вычислений требуется увеличить значение переменной X на I. Это действие реализуется с помощью следующего оператора присваивания:

```
10 LET X=X+1
```

Здесь новое значение X получается в результате прибавления единицы к старому значению X. Этот прием часто используется в программировании для наращивания значений переменной с некоторым шагом.

Служебное слово *LET* при вводе текста программы в ЭВМ может быть опущено. В некоторых компьютерах оно добавляется автоматически после занесения оператора в память машины. Поэтому при выводе текста программы на носителе, например при вызове программы на экран дисплея с помощью директивы *LIST*, слово *LET* появится всюду, где оно было при вводе опущено.

Рассмотрим некоторые примеры:

Математическая запись	Запись на языке БЕЙСИК
$D=b^2-4ac$	`10 D=B^2-4*A*C`
$P_3=aX^3+a_1X^2+a_2X+a_3$	`20 P3=A(0)*X^3+A(1)*X^2+A(2)*X+A(3)`
$y=\frac{\ln\sqrt{x}}{\lvert x^2+y^2\rvert}$	`30 Y=LOG(SQR(X))/ABS(X^2+Y^2)`
$t=\frac{\varphi(z)}{\sin\alpha\cdot\ln z}$	`40 T=(F(Z))/SIN(A)/LOG(Z)`

2.2. Простейший ввод и вывод информации

Программы редко составляются для решения некоторой конкретной задачи с жестко заданными числовыми значениями обрабатываемых данных. Обычно программа составляется в некотором обобщенном виде, когда в реализуемые согласно алгоритму арифметические выражения входят переменные, а конкретные числовые значения им присваиваются во время выполнения программы. Эта возможность обеспечивается с помощью операторов ввода данных. Рассмотрим простейший из них - оператор *INPUT* (другие возможности ввода данных будут рассмотрены в разд. 2.4).

Оператор ввода данных. Для присваивания переменным значений, вводимых во время исполнения программы по ее запросу с клавиатуры дисплея, служит оператор *INPUT* , который имеет следующий формат:

INPUT <список переменных>

<Список переменных> содержит разделенные запятыми имена переменных (простых или индексированных), которым во время работы программы будут присвоены конкретные числовые значения.

Встретив в программе оператор *INPUT* указанного формата, ЭВМ делает паузу, выводит знак "?" (знак вопроса) и ждет. Для продолжения работы программы пользователь должен набрать на клавиатуре вводимые числа, разделяя их запятыми, и после набора последнего числа нажать клавишу "Ввод". Первое набранное на клавиатуре число будет присвоено первой переменной, указанной в списке оператора *INPUT* , второе число будет присвоено второй переменной из списка и т.д. Если, например, необходимо ввести числовые значения для *X* и *Y* , нужно в программе вставить оператор

20 INPUT X,Y

до первого оператора, использующего любую из этих переменных. Когда встречается этот оператор, машина выводит на экран дисплея вопросительный знак и ждет. Пользователь вводит два числа, разделенные запятой, нажимает клавишу "Ввод", и машина продолжает выполнять оставшуюся часть программы.

Кроме имен переменных, в оператор *INPUT* может быть включен произвольный текст, заключенный в апострофы и предшествующий списку переменных. Формат оператора:

INPUT "<произвольный текст>" [;] <список переменных>

Заключенный в апострофы текст и следующая за ним переменная запятой обычно не разделяются. (В ПК *YAMAHA* здесь обязательна точка с запятой.) При выполнении оператора *INPUT* приведенного выше формата машина выдает на печать текст (уже без апострофов) и приостанавливает работу программы. После этого пользователь должен набрать на клавиатуре вводимые числа, как и в случае формата оператора *INPUT* , без поясняющего текста. Обычно заключенный в апострофы текст представляет собой подсказку о том, какие переменные ожидают вводимые с клавиатуры числовые значения. Например, оператор *30 INPUT* "Введите коэффициенты и свободный член" *A* , *B* , *C* выведет на экран дисплея текст

после чего машина остановится. Набранные с клавиатуры три числа будут присвоены переменным *A*, *B*, *C* в соответствии со списком ввода.

Как и оператор *LET*, оператор *INPUT* определяет новые переменные программы, выделяя для них место в памяти ЭВМ.

Оператор вывода. Вывод (печать) результатов осуществляется в языке БЕЙСИК с помощью оператора *PRINT*, имеющего формат:

PRINT <список вывода>

Элементы <списка вывода> могут содержать переменные, арифметические выражения (в частности, числа), заключенные в апострофы текста, а также служебные слова. В качестве разделителей в выводном списке оператора используются запятая и точка с запятой, которые служат одновременно для управления печатью.

Если список вывода содержит имя переменной, то при выполнении оператора *PRINT* произойдет вывод числового значения этой переменной, но само имя не выводится. Например, в результате выполнения оператора 2Ø *PRINT A;B,C* на экран дисплея может быть выдана (в зависимости от текущих значений переменных) строка вида

95.7382Ø35 - I39.ØØØØØØØ .95I83Ø3

Список вывода оператора *PRINT* может содержать и арифметические выражения, например:

*30 PRINT B*B-4*A*C, XI-SIN(X(1))*

В этом случае оператор *PRINT* производит все предусмотренные арифметическим выражением вычисления и выдает полученные числовые результаты. Разумеется, все входящие в арифметическое выражение переменные (простые и индексированные) должны быть определены в операторах, предшествующих оператору вывода.

Весьма удобной особенностью оператора *PRINT* является то, что он может использоваться для выдачи различной текстовой информации: пояснений, таблиц, графиков и др. Для этого подлежащую выводу информацию достаточно включить в список вывода оператора *PRINT*, предварительно заключив ее в кавычки. Пусть, например, текущее значение переменной *Z* равно I07. Тогда выполнение оператора

20 PRINT "максимальное значение =" Z ;

повлечет выдачу на экран дисплея такой строки:

МАКСИМАЛЬНОЕ ЗНАЧЕНИЕ = 107

Рассмотрим несложную вычислительную программу.

Пример 2.1. Пусть v_0 , a - соответственно начальная скорость и ускорение движения тела; t - время движения. Путь, пройденный телом, выражается формулой

$$S = v_0 t + \frac{a t^2}{2}.$$

Программа для вычисления значений S в зависимости от заданных значений a , v_0 , t может иметь вид

```
10 REM ПУТЬ
20 INPUT V0,A,T
30 S=V0+AXT/2
40 S=SXT
50 PRINT"ПРОЙДЕННЫЙ ПУТЬ=";S
60 END
```

Программа состоит из шести строк. Первая из них содержит неисполнимый оператор *REM* с комментарием, облегчающим чтение листинга программы (текста, выданного машиной). В общем случае оператор имеет вид:

REM <произвольный текст> ,

где <произвольный текст> является собственно комментарием и в кавычки не заключается. Оператор не исполняется машиной и может размещаться в любом месте программы.

Во второй строке программы расположен оператор ввода *INPUT*, с помощью которого переменным *V0*, *A* и *T* присваиваются конкретные числовые значения, вводимые пользователем с клавиатуры дисплея во время выполнения этого оператора.

В третьей и четвертой строках находятся операторы присваивания, которые реализуют вычислительную формулу:

$$S = (V_0 + \frac{at}{2}) t.$$

Обратите внимание, что значение выражения в скобках сначала присваивается переменной *S* (строка 3Ø), а затем в операторе с номером 4Ø промежуточное значение переменной *S* умножается на *T* и результат снова присваивается переменной *S* как окончательный. И, наконец, в строке с номером 5Ø расположен оператор вывода *PRINT*, тело которого составлено таким образом, что машина выведет не только само значение *S*, но и предшествующее ему пояснение. Если, например, *VØ* = 5.3, *A* = 2.6, *T* = 3, то машина выведет следующий результат:

ПРОЙДЕННЫЙ ПУТЬ = 27.6

Завершает программу оператор *END*. Использование этого оператора в конце текста программы не обязательно, так как выполнение ее автоматически прекращается при достижении последнего оператора.

Рассмотрим, как осуществляется простейшее управление выводом информации на экран дисплея (или на печать). Наличие запятой между элементами списка вывода оператора *PRINT* обеспечивает так называемую зонную печать, суть которой состоит в следующем. Вся строка экрана дисплея условно разделена на зоны (ширина зоны зависит от типа компьютера). Каждая запятая в выводном списке оператора *PRINT* вызывает печать в следующей зоне, а если свободные зоны строки исчерпались - в следующей строке. Две запятые, стоящие рядом, обусловливают пропуск зоны вывода.

Пример. Пусть *A* = 3.75, *B* = 9.3, *C* = 0.7. Оператор вывода *1Ø PRINT A,"сумма", B+C* обеспечивает выдачу следующей строки (в ПК *YAMAHA*):

Номер позиции в строке	I	I5
Выводимая информация	3.75 I Ø.ØØ	Сумма

Если между элементами списка вывода оператора *PRINT* стоит точка с запятой (";"), то информация выводится в плотном формате: каждая точка с запятой вызывает смещение при выводе очередного элемента вправо на одну позицию относительно ранее выведенного элемента списка.

Например, если $A = 3.75$, $B = 9.3$, $C = 0.7$, то оператор вывода

```
10 PRINT A ; "сумма" ; B+C
```

выдаст строку на экран

```
 3.75  С У М М А  10 . 00
```

(другие способы управления выводом будут рассмотрены позже).

Упражнения

1. Составить программу вычисления периода T и частоты E колебания маятника в зависимости от его длины ℓ. Известно, что период (время одного полного колебания) маятника вычисляется по формуле

$$T = 2\pi\sqrt{\frac{\ell}{g}},$$

где T – время, с; ℓ – длина маятника, м; g – ускорение свободного падения. Частота

$$E = 1/T.$$

2. В треугольнике известны длины сторон a и b и угол между ними α. Составить программу вычисления длины третьей стороны C, площади треугольника S и радиуса описанной окружности R.

3. Радиус окружности, вписанной в равносторонний треугольник, равен R. Составить программу вычисления длин сторон, высоты и площади треугольника.

4. Составить программу вычисления площади треугольника по формуле Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где p – полупериметр треугольника.

2.3. Операторы управления программой

Используя лишь оператор присваивания и, быть может, операторы ввода-вывода, на ЭВМ удается реализовать в виде программы только линейные алгоритмы, т.е. такие, в которых операторы выполняются последовательно друг за другом в порядке их естественного расположения

(в порядке возрастания номеров строк) и этот порядок не зависит от конкретных значений обрабатываемых данных.

Если требуется программно изменить этот естественный порядок выполнения операторов (осуществить обход некоторой группы подряд расположенных операторов, организовать повторное выполнение некоторой последовательности операторов), то используют операторы управления программой. Эта необходимость возникает при реализации разветвляющихся и циклических алгоритмов [8].

К операторам управления относятся операторы условного и безусловного переходов, а также оператор цикла.

Операторы безусловного перехода. Общий вид оператора:

GOTO N

где *GOTO* - служебное слово; *N* - номер строки (целое число без знака).

Действие этого оператора заключается в безусловном переходе к строке с указанным номером. Если при этом осуществляется переход на неисполнимый или отсутствующий оператор, то управление передается ближайшему по порядку исполнимому оператору. Например, оператор *GOTO* 120 предписывает программный переход к строке с номером 120.

Обратимся к примеру 2.1. Пусть при фиксированных значениях *V0*= 113, *A* = 19 необходимо получить ряд значений *S* при последовательных значениях *T* = 12, 24, 36, 48. Очевидно, что для этой цели можно было бы воспользоваться программой примера 2.1, вводя ее в память заново при каждом новом наборе исходных данных. Однако эту программу нетрудно видоизменить так, чтобы ее без повторного ввода можно было бы использовать для многократных вычислений. Прежде всего, оператор ввода (строка с номером 20) разобъем на два оператора:

20 INPUT V0,A

25 INPUT T

Теперь, для многократного вычисления значения *S* при указанных последовательных значениях *T*, достаточно сделать так, чтобы после каждого очередного выполнения оператора вывода (строка 50) снова и снова выполнялся оператор ввода величины *T*. Это может быть достигнуто заменой оператора *END* (строка 60) оператором безусловного перехода *GOTO* 25. После указанных преобразований программа будет иметь вид

```
10 REM ПУТИ
20 INPUT V0,A
25 INPUT T
30 S=V0+AXT/2
40 S=SXT
50 PRINT "ПРОЙДЕННЫЙ ПУТЬ";S
60 GOTO 25
```

Протокол работы данной программы для указанных числовых значений исходных данных:

```
? 113, 19
? 12
Пройденный путь   2724.000
? 24
Пройденный путь   8184.000
? 36
Пройденный путь   1638.000
? 48
Пройденный путь  27312.000
```

Обратите внимание, что после запуска программы оператор

20 INPUT V0,A

выполняется только один раз, в то время как операторы 25 – 60 выполняются многократно.

Оператор условного перехода. Для программирования разветвляющихся алгоритмов в языке БЕЙСИК имеется оператор условного перехода (или, коротко, оператор *IF*). Напомним, что разветвляющимися называют алгоритмы, в которых последовательность обработки данных зависит от истинности или ложности некоторых проверяемых условий.

Существуют несколько форм записи оператора условного перехода. Наиболее распространенной (назовем ее первой формой) является следующая:

IF <условие> *THEN* <номер строки>

Действует этот оператор так: если указанное после служебного слова *IF* условие при текущих значениях входящих в него переменных истинно, то осуществляется переход к строке, номер которой указан после служебного слова *THEN*; в противном случае произойдет переход к следующей по порядку строке (но не к следующему оператору, если в строке после *IF*-оператора имеются другие операторы, разделенные двоеточиями).

Пример:

```
10 IF X*X+Y*Y <=R^2 THEN 30
20 PRINT "условие не удовлетворяется"
30 .....
```

Этот оператор предписывает машине выполнить такие действия:

а) вычислить значения выражений X^2+Y^2 и R^2 при текущих значениях X, Y, R;

б) если для вычисленных значений отношение $X^2+Y^2 \leq R^2$ удовлетворяется, то перейти к выполнению строки с номером 30; иначе – к строке с номером 20, оператор которой выведет на экран дисплея текст:

УСЛОВИЕ НЕ УДОВЛЕТВОРЯЕТСЯ

В приведенном формате оператора *IF* служебное слово *THEN* можно заменять словом *GOTO*:

IF <условие> *GOTO* <номер строки>

В рассматриваемом варианте интерпретатора языка БЕЙСИК допускается следующая интересная и важная для приложений форма *IF* -оператора (вторая форма)*:

IF <условие> *THEN* <оператор>

т.е. после служебного слова *THEN* допускается любой оператор БЕЙСИКа, в том числе и другой оператор *IF*. Если проверяемое условие истинно, то выполняется упомянутый оператор, а вслед за ним последовательно реализуются другие операторы в этой строке, если они есть. Если же проверяемое условие не удовлетворяется, то указанный после *THEN* оператор не выполняется и осуществляется переход к следующей по порядку строке программы. Последнее следует учитывать при записи в одной строке с оператором условного перехода других операторов. А именно, если первым оператором в строке является *IF* -оператор и записанное в нем условие не удовлетворяется, то пропускаются все операторы этой строки, следующие после *THEN*, и осуществляется, как было сказано, переход к следующей строке.

---

* В современных реализациях БЕЙСИКа допускается полная условная конструкция:

IF <условие> *THEN* <операторы> *ELSE* <операторы>

Например, во фрагменте программы

```
10 A=0: B=5
50 IF A<>0 THEN B=0: GOTO 70
60 B=B+3
70 PRINT B
```

программа выведет в качестве значения переменной B число 8, а не число 5, как может показаться вначале.

Прежде чем перейти к составлению разветвляющихся программ, рассмотрим общие приемы программирования базовой структуры РАЗВИЛКА средствами языка БЕЙСИК. Понимание общих принципов программной реализации базовых алгоритмических структур облегчит затем составление более сложных программ, включающих в себя базовые структуры как составные части.

Полная условная конструкция. Программирование этой базовой структуры (рис. 2.1) проследим подробно на примере. Рассмотрение конкретного примера не ограничивает общности рассуждений, так как выполняемые в этом случае действия типичны для полной условной конструкции и в общем случае.

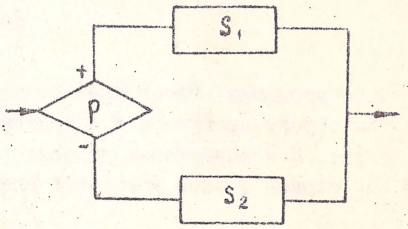


Рис. 2.1

Пример 2.2. Функция задана графически (рис. 2.2). Составить программу вычисления значения этой функции в зависимости от значения аргумента x.

Из графика следует, что

$$y=\begin{cases} 0, & \text{если } |x|>1, \\ \sqrt{1-x^2}, & \text{если } |x|\le 1. \end{cases}$$

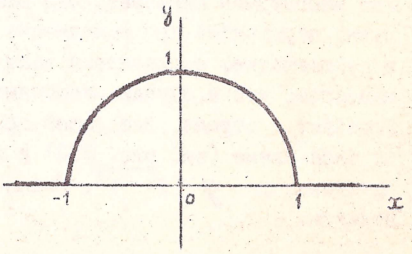


Рис. 2.2

Блок-схема алгоритма вычисления этой функции приведена на рис. 2.3. Так как выбор вычислительной формулы зависит от истинности или ложности некоторого условия $(|x|>1)$, то это разветвляющийся алгоритм. Его основу составляет базовая структура РАЗВИЛКА, помещенная в пунктирный

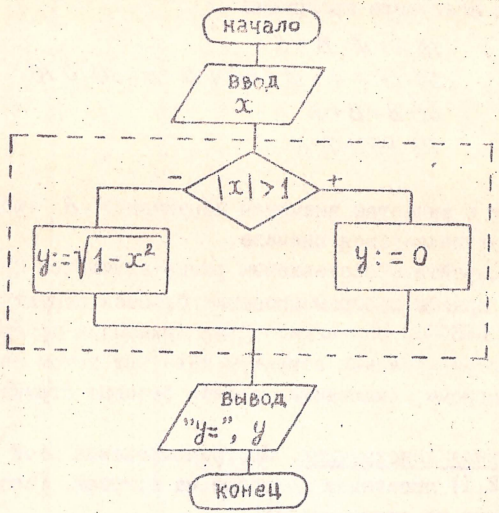


Рис. 2.3

прямоугольник. Рассмотрим подробно ее программную реализацию. Составим строку программы с оператором *IF*, проверяющим в соответствии с рис. 2.3 выполнение условия $|X| > 1$. Сначала воспользуемся первой формой записи условного оператора:

```
20 IF ABS(X)>1 THEN...
```

где многоточие означает, что пока не известен номер строки, к которой нужно переходить при выполнении условия $|X| > 1$. В то же время в соответствии с правилами исполнения оператора *IF* доподлинно известно, что в случае невыполнения проверяемого условия машина перейдет к строке, следующей сразу же за оператором *IF*, а согласно блок-схеме (см. рис. 2.3) в этом случае должен выполняться блок присваивания $y:=\sqrt{1-x^2}$. Именно эту логическую ветвь следует продолжать:

```
20 IF ABS(X)>1 THEN...
30 Y=SQR(1-X^2)
```

Здесь логическая ветвь, соответствующая условию $|X| \leq 1$, реализуется в виде одной строки БЕЙСИК-программы. В общем же случае как альтернатива "истина", так и альтернатива "ложь" структуры РАЗВИЛКА могут быть реализованы в виде нескольких строк программы.

Согласно схеме структуры РАЗВИЛКА (см. рис. 2.I) после выполнения последовательности операторов, соответствующих альтернативе "ложь", должен происходить безусловный переход к общему выходу из нее; поставим оператор GOTO в той же строке 3Ø (либо в следующей строке):

```
2Ø IF ABS(X)>I THEN...
3Ø Y=SQR(I-X^2):GOTO...
```

Многоточие после GOTO означает, что номер строки выхода из структуры РАЗВИЛКА также пока неизвестен. Замечаем теперь, что для продолжения программы, когда условие /X/ > 1 истинно, может быть использована строка с номером 4Ø - это позволит исключить многоточие в строке 2Ø:

```
2Ø IF ABS(X)>I THEN 4Ø
3Ø Y=SQR(I-X^2):GOTO...
4Ø Y=Ø
```

Теперь совершенно ясно, что выходом из структуры РАЗВИЛКА (общим для обеих ветвей!) будет строка с номером 5Ø. Этим номером и следует дополнить оператор GOTO в строке 3Ø. Окончательная запись программы, реализующей алгоритм поставленной выше задачи, примет вид:

```
1Ø INPUT X
15 PRINT "X=";X
2Ø IF ABS(X)>1 THEN 4Ø
3Ø Y=SQR(1-X*X):GOTO 5Ø
4Ø Y=Ø
5Ø PRINT "Y=";Y
6Ø END
```

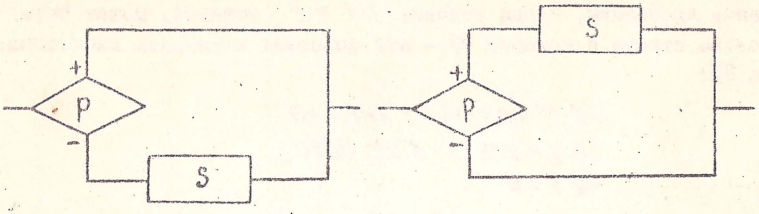
Для удобства сравнения с блок-схемой (см. рис. 2.3) здесь реализация базовой структуры РАЗВИЛКА также помещена в пунктирный прямоугольник.

Воспользуемся второй формой записи условного оператора для решения рассматриваемой задачи:

```
1Ø INPUT X
15 PRINT "X=";X
2Ø IF ABS(X)>1 THEN Y=Ø:GOTO 4Ø
3Ø Y=SQR(1-X*X)
4Ø PRINT "Y=";Y
5Ø END
```

Совершенно ясно, что эта форма оператора *IF* более предпочтительна по сравнению с первой, когда число операторов, реализующих альтернативу "истина", невелико (помещается в одной строке с оператором *IF*). Обычно при использовании этого оператора удается сократить количество операторов программы и избавиться от излишних операторов безусловного перехода, использование которых следует по возможности ограничивать.

Неполная условная конструкция. В неполной условной конструкции исполнимый блок *S* может быть расположен как на ветви "+" (рис. 2.4,а), так и на ветви "-" (рис. 2.4,б).



а Рис. 2.4 *б*

Наиболее просто и естественно программируется случай, когда блок *S* находится на линии "-", с помощью первой формы *IF* -оператора:

```
2Ø IF P THEN 4Ø
3Ø [ S ]
4Ø Выход
```

Если же блок *S* расположен на линии "+" и число операторов, реализующих блок *S*, невелико, то целесообразно использовать вторую форму:

```
2Ø IF P THEN [ S ]
3Ø Выход
```

Когда из-за большого числа операторов, соответствующих блоку *S*, воспользоваться этой формой оператора не удается, то целесообразно в качестве условия в операторе *IF* взять не *P*, а его отрицание (обозначим его $\bar{P}$):

```
1Ø IF P̄ THEN 4Ø
3Ø [ S ]
4Ø Выход
```

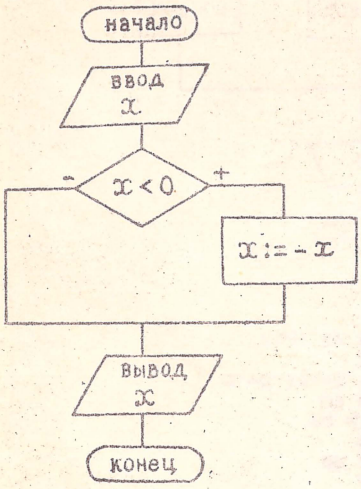
Пример 2.3. Составить программу вычисления функции

$$y = |x|$$

Напомним, что по определению абсолютной величины числа X имеем:

$$y = \begin{cases} X, & \text{если } X \geq 0 \\ -X, & \text{если } X < 0 \end{cases}$$

Соответствующая блок-схема и программа показаны на рис. 2.5.



а

```
5  REM МОДУЛЬ
10 INPUT X
20 PRINT"X=";X
30 IF X<0 THEN X=-X
40 PRINT"МОДУЛЬ X=";X
50 END
```

б

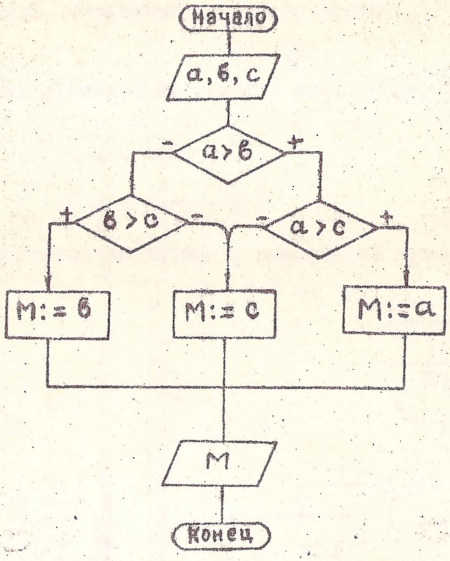
Рис. 2.5

Обратите внимание, что в этом примере блок 5 неполной условной конструкции, о которой говорилось, сводится к изменению знака X, если $X < 0$; в противном случае никакие действия не выполняются и управление сразу передается на общий для обеих ветвей выход (оператор 40).

Пример 2.4. Составить программу поиска большего из трех чисел.

Блок-схема алгоритма изображена на рис. 2.6,а, а программа – на рис. 2.6,б.

Тело оператора *PRINT* организовано таким образом, что на экран будет выведен полученный результат в виде $MAX(A,B,C) = M$, где A, B, C – конкретные числовые значения, которые присвоены переменным A, B, C с помощью оператора *INPUT*, а M – найденное максимальное значение.



а

```
10  REM БОЛЬШЕЕ ИЗ ТРЕХ
20  INPUT A,B,C
25  PRINT"A="A;"B="B;"C="C
30  IF A>B THEN 60
40  IF B<=C THEN 80
50  M=B:GOTO 90
60  IF A<C THEN 80
70  M=A:GOTO 90
80  M=C
90  PRINT "MAX(";"A","B","C")=";M
100 END
```

РЕЗУЛЬТАТЫ РАБОТЫ:

```
A= 11 B=-88 C= 13
MAX( 11 ,-88 , 13 )= 13
```

б

Рис. 2.6

Пример 2.5. Составить программу, определяющую, могут ли три наугад взятые числа быть длинами сторон треугольника. При положительном ответе программа должна выяснить тип треугольника (равносторонний, разносторонний, равнобедренный). Блок-схема изображена на рис. 2.7.

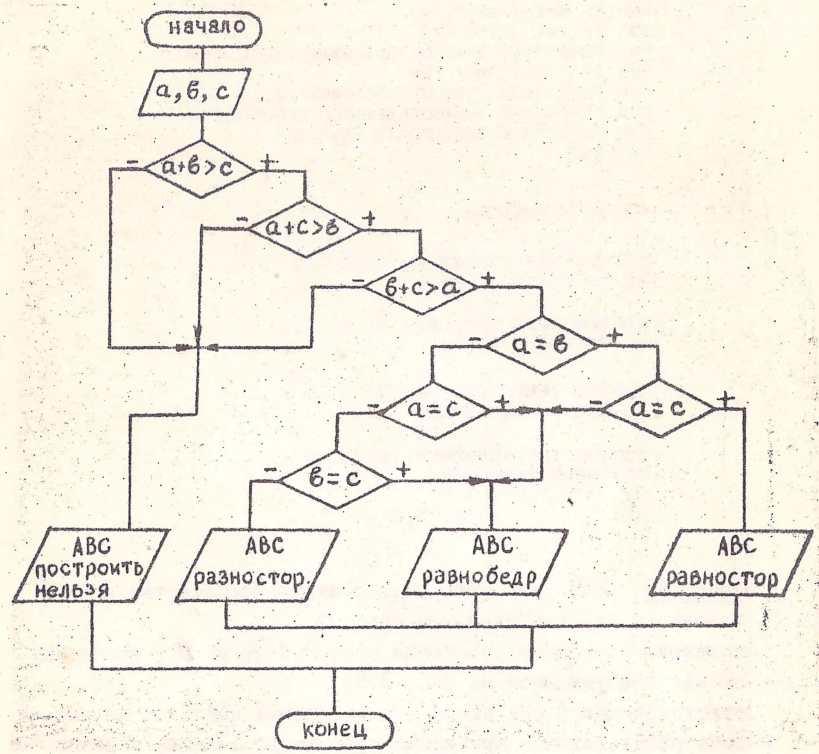


Рис. 2.7

При составлении этой блок-схемы мы несколько отступили от требований структурной алгоритмизации с целью сокращения размеров блок-схемы за счет устранения возможных повторений одинаковых блоков.

Программа, реализующая приведенный на рис. 2.7 алгоритм, может иметь вид (рис. 2.8):

```
10  REM ТРЕУГОЛЬНИК
20  INPUT A,B,C
30  PRINT"СТОРОНЫ ТРЕУГОЛЬНИКА:"A;B;C
40  IF (A+B)<=C THEN 130
50  IF (A+C)<=B THEN 130
60  IF (B+C)<=A THEN 130
70  IF A=B THEN 100
80  IF A=C THEN 120
85  IF B=C THEN 120
90  PRINT"ABC-РАЗНОСТОРОННИЙ":GOTO 140
100 IF A<>C THEN 120
110 PRINT"ABC-РАВНОСТОРОННИЙ":GOTO 140
120 PRINT"ABC-РАВНОБЕДРЕННЫЙ":GOTO 140
130 PRINT"ABC-ПОСТРОИТЬ НЕЛЬЗЯ"
140 END


ПРОТОКОЛ РАБОТЫ:

СТОРОНЫ ТРЕУГОЛЬНИКА: 3  4  5
ABC-РАЗНОСТОРОННИЙ

СТОРОНЫ ТРЕУГОЛЬНИКА: 1  2  2
ABC-РАВНОБЕДРЕННЫЙ

СТОРОНЫ ТРЕУГОЛЬНИКА: 3  3  6
ABC-ПОСТРОИТЬ НЕЛЬЗЯ

СТОРОНЫ ТРЕУГОЛЬНИКА: 6  6  6
ABC-РАВНОСТОРОННИЙ
```

Рис. 2.8

Пример 2.6. Составить программу решения неравенства $ax < b$ при произвольных действительных числах a, b.

Анализируя возможные сочетания значений a и b, приходим к блок-схеме, изображенной на рис. 2.9.

Соответствующая этому алгоритму программа показана на рис. 2.10.

Выдача результатов в программе организована таким образом, что сначала выводится само решаемое неравенство /строка 30/, а затем в следующей строке - результат его решения:

```
РЕЗУЛЬТАТЫ РАБОТЫ:


НЕРАВЕНСТВО 0 X< 0
РЕШЕНИЙ НЕТ
НЕРАВЕНСТВО 0 X< 3.5
X-ЛЮБОЕ ЧИСЛО
НЕРАВЕНСТВО-2 X< 5
X>-2.5
НЕРАВЕНСТВО 2 X< 5
X< 2.5
```

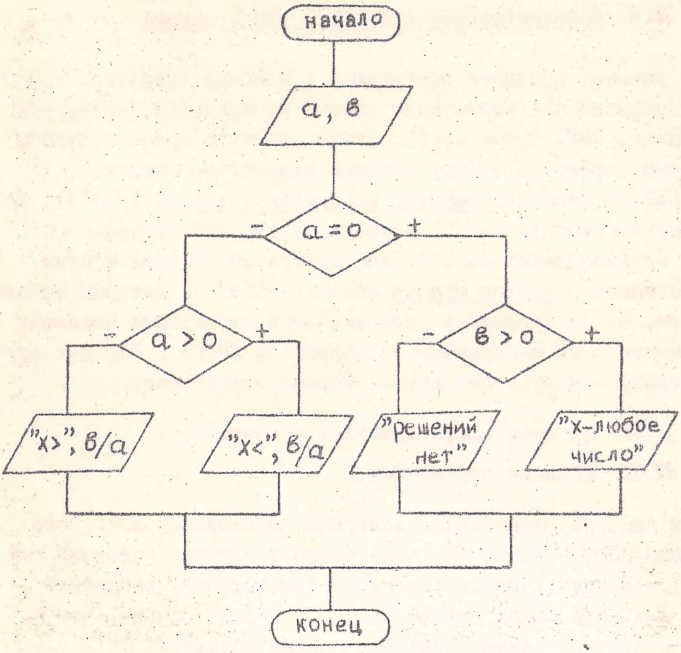


Рис. 2.9

```
10 REM НЕРАВЕНСТВО AX<B
20 INPUT A,B
30 PRINT "НЕРАВЕНСТВО";A;"X<";B
40 IF A=0 THEN 70
50 IF A>0 THEN PRINT "X<"B/A:GOTO 90
60 PRINT "X>"B/A:GOTO 90
70 IF B>0 THEN PRINT "X-ЛЮБОЕ ЧИСЛО":GOTO 90
80 PRINT "РЕШЕНИЙ НЕТ"
90 GOTO 20
```

Рис. 2.10

2.4. Дополнительные операторы ввода данных

Числовые данные, вводимые программой с помощью оператора *INPUT*, при повторном запуске программы надо заново набирать на клавиатуре и вводить в память ЭВМ. Кроме того, требуется много времени, чтобы ввести с помощью оператора *INPUT* большое количество данных.

В языке БЕЙСИК имеется оператор определения данных (*DATA*), который позволяет заготовить обрабатываемые данные заблаговременно и расположить их непосредственно в теле программы. В паре с этим оператором выступает оператор чтения данных (*READ*), который используется для того, чтобы присвоить ряду заданных переменных числовые значения, автоматически выбираемые из оператора *DATA*. Эти два оператора не используются друг без друга. Формат операторов:

DATA <список арифметических выражений>

READ <список переменных>

Элементы обоих списков разделяются запятыми. В качестве элементов списка оператора *DATA* выступают чаще всего константы (частный случай выражений). В случае использования арифметических выражений применяемые в них переменные должны быть определены раньше, чем будет использован соответствующий элемент списка оператора *DATA*.

Так, если в программе записан оператор

DATA 5, 4.37, 2.71E+2, 97, 3.1415

то в результате выполнения оператора ввода

READ A,B,C,D

произойдут следующие присваивания: $A = 5$, $B = 4.37$, $C = 271$, $D = 97$.

Сами по себе операторы *DATA* никакого действия не производят и могут располагаться в любом месте программы. В программе может быть несколько операторов *DATA* и *READ* и число их может не совпадать, однако количество констант, вводимых в программу операторами *DATA*, должно быть по крайней мере не меньше, чем количество переменных, содержащихся в операторах *READ*. Перед выполнением программы машина выбирает из всех операторов *DATA* числовые данные в том порядке, в каком они расположены в программе, и образует общий блок данных. Оператор чтения вызывает присваивание переменным, перечисленным в его списке ввода, очередных доступных чисел из общего блока данных.

Оператор *DATA* должен быть единственным или последним оператором в строке.

Каждый раз, когда в любом месте программы встречается оператор *READ*, общий блок данных поставляет ему очередное число. Каждый очередной оператор программы *READ* производит выборку величин из общего блока, начиная присвоение с первого неиспользованного при предыдущем чтении значения общего блока.

Пример.

```
2Ø DATA 5.2,-.15,2.85
3Ø DATA 8.25,.37E-5
4Ø DATA .45,-1.5E3,12
5Ø READ A,B
6Ø READ C,D,E
7Ø READ X,Y(5)
```

Здесь номера строк операторов выбраны произвольно. При выполнении программы оператора *DATA* строки 2Ø, 3Ø, 4Ø игнорируются до тех пор, пока не встретится исполнимый оператор *READ*. Эти три оператора задают общий блок данных, состоящий из восьми чисел, расположенных в такой последовательности:

5.2; -0.15; 2.85; 8.25; $0.37 \cdot 10^{-5}$; 0.45; $-1.5 \cdot 10^{3}$; 12

Первый оператор *READ* выполнит присваивание: $A = 5.2$; $B = -0.15$; второй оператор *READ*, продолжая последовательную выборку из общего блока данных, присвоит переменным C, D, E соответственно значения 2.85; 8.25; $0.37 \cdot 10^{-5}$. Наконец, последний оператор *READ* (строка 7Ø) выполнит присваивание: $X = 0.45$; $Y(5) = -1.5 \cdot 10^{3}$. В последнем случае значение присвоено индексированной переменной.

При необходимости пропуска отдельных элементов блока данных можно использовать некоторую фиктивную переменную, повторяя ее идентификатор в списке ввода требуемое число раз. Пусть, например, блок данных содержит последовательность чисел: 3, 1.2; 0.37; 8.9; -7.3. Требуется присвоить переменным A и B числовые значения 8.9 и -7.3 соответственно:

```
1Ø DATA 3, 1.2, .37, 8.9, -7.3
2Ø READ W,W,W,A,B
```

При выполнении последнего оператора будут обеспечены следующие присвоения: W = 3, W = I.2, W = 0.37, A = 8.9, B = 7.3, что и требовалось. Здесь W играет роль фиктивной переменной, которая в дальнейшем нигде не используется.

После выборки из блока данных последнего числа он считается исчерпанным. Обращение к опустошенному блоку данных считается ошибкой и приводит к снятию программы. Если возникает необходимость в повторном использовании блока данных, его предварительно следует восстановить с помощью оператора RESTORE . Этот оператор не имеет параметров.

Пример.

```
10 DATA 3, I.2, .37, 8.9, -7.3
20 READ A,B, C(5), X, Z(2,3)
   .....................
60 RESTORE
70 READ X,Y,Z,U(3),W
```

После первой выборки данных из блока оператором READ (строка 20) он окажется полностью исчерпанным. Оператором RESTORE блок данных восстанавливается в первоначальное состояние. Второй оператор READ (строка 70) снова обращается к началу блока данных, присваивая последовательно: X = 3, Y = I,2, Z = 0,37 и т.д.

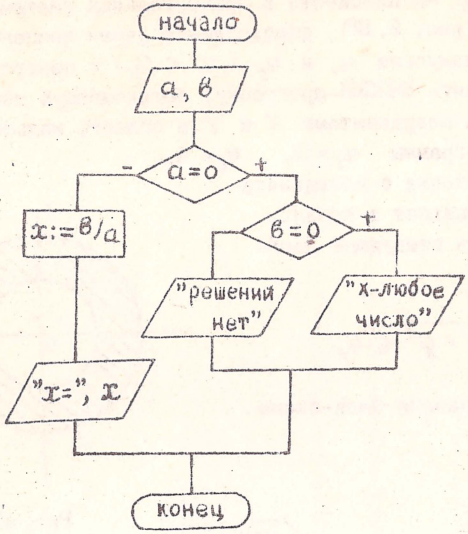
Рассмотрим несколько примеров.

Пример 2.7. Составить программу решения уравнения $aX=B$ для произвольных значений числовых параметров a и B.

Блок схема алгоритма изображена на рис. 2.II,а, а соответствующая программа - на рис. 2.II,б.

Приведенная на рис. 2.II,б программа составлена для трех пар значений a, B иллюстрирующих работу алгоритма для трех принципиально отличимых случаев соотношения между параметрами. Операторы DATA - READ особенно удобны при отладке программ (проверке их правильности для всех логически возможных случаев).

В приведенной реализации программа введет (благодаря наличию оператора GOTO 30 в строке 100) последовательные пары чисел (0, 0); (0, 5); (3, I2.6), после чего блок данных будет исчерпан. Программа завершается аварийно с выдачей соответствующей индикации об ошибке, хотя в данном случае авария была заранее предусмотрена.



а

```
10  REM УРАВНЕНИЕ AX=B
20  DATA 0,0,0,5,3,12.6
30  READ A,B
40  IF A=0 THEN 70
50  X=B/A
60  PRINT"X=";X:GOTO 100
70  IF B=0 THEN 90
80  PRINT"РЕШЕНИЙ НЕТ":GOTO 100
90  PRINT"X-ЛЮБОЕ ЧИСЛО"
100 GOTO 30
```

ПРОТОКОЛ РАБОТЫ:

```
X-ЛЮБОЕ ЧИСЛО
РЕШЕНИЙ НЕТ
X= 4.2
```

б

Рис. 2.II

Пример 2.8. На плоскости в прямоугольной системе координат задано кольцо (рис. 2.I2), определяемое двумя концентрическими окружностями радиусами r_1 и r_2 ($r_1 < r_2$) с центром в начале координат. Составить БЕЙСИК-программу, определяющую попадание точки с произвольными координатами X и Y в область кольца. Принять для отладки программы $r_1 = 2$, $r_2 = 4$.

Очевидно, точка с координатами (X, Y) попадает в кольцо, если выполняется следующее двойное неравенство:

$$r_1 \le \sqrt{X^2 + Y^2} \le r_2 .$$

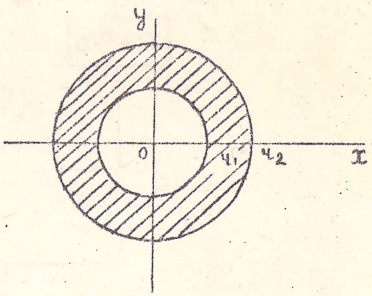


Рис. 2.I2

На рис. 2.I3 показана блок-схема алгоритма.

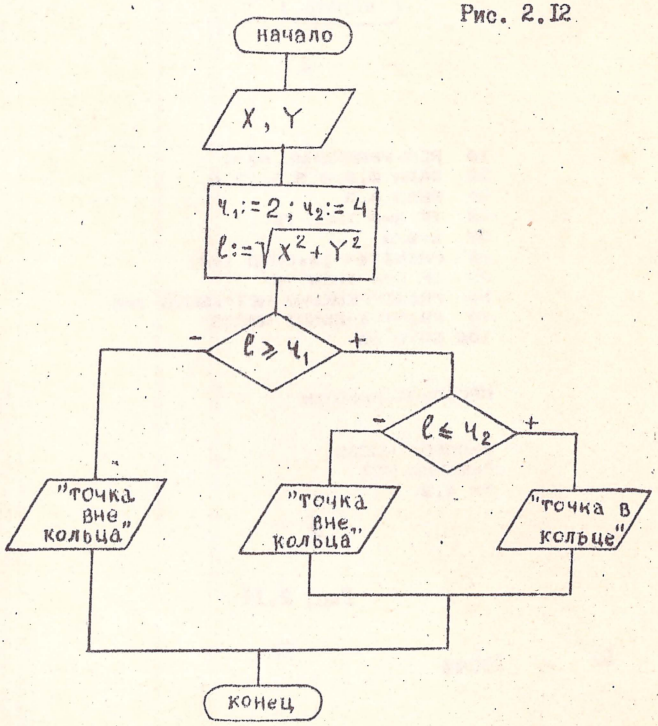


Рис. 2.I3

Соответствующая этой блок-схеме программа может иметь вид (рис. 2.I4):

```
10  REM ПОПАДИ В КОЛЬЦО
20  DATA 0,1,2,3,4,5
30  R1=2:R2=4
40  READ X,Y
50  L=SQR(X*X+Y*Y)
60  IF L<R1 THEN 100
70  IF L>R2 THEN 100
80  PRINT "ТОЧКА ("X;Y") В КОЛЬЦЕ"
90  GOTO 40
100 PRINT "ТОЧКА ("X;Y") ВНЕ КОЛЬЦА"
110 GOTO 40

ПРОТОКОЛ РАБОТЫ:

ТОЧКА ( 0  1 ) ВНЕ КОЛЬЦА
ТОЧКА ( 2  3 ) В КОЛЬЦЕ
ТОЧКА ( 4  5 ) ВНЕ КОЛЬЦА
```

Рис. 2.I4

Для отладки программы с целью проверки правильности ее функционирования для всех логически возможных случаев достаточно последний оператор использовать в виде

```
II0   GOTO 40
```

Программа в этом случае хотя и завершится аварийно с выдачей соответствующего сообщения, однако поставленную задачу выполнит. Аварийное завершение, как и в предыдущем примере, вызвано обращением оператора ввода

```
40   READ X,Y
```

к опустошенному блоку данных после трехкратного ввода пар чисел: (0,I); (2,3); (4,5).

Протокол работы приведен сразу после текста программы (см. рис. 2.I4).

Пример 2.9. На рис. 2.I5 изображена программа, реализующая игру Баше. Правила игры приведены в тексте программы и выдаются по запросу играющего с машиной. Протокол всей игры приведен после текста программы. Первый ход делает играющий с машиной и всегда проигрывает.

```
10  REM ИГРА ВАШЕ
20  INPUT"НУЖНЫ ПРАВИЛА ИГРЫ?(1/0)";K
30  IF K=0 THEN 60
40  PRINT"ДАНЫ N ПРЕДМЕТОВ.ЗА ХОД "
50  PRINT"МОЖНО ВЗЯТЬ ОТ 1 ДО 5 ПРЕДМЕ"
60  PRINT"ТОВ.ВЫИГРЫВАЕТ ВЗЯВШИЙ ПОСЛЕ"
70  PRINT" ДНИЙ ПРЕДМЕТ"
80  S=30
90  PRINT"Н А Ч А Л О";S;" ПРЕДМЕТОВ"
100 PRINT"ВАШ ХОД";:INPUT K:PRINT K
110 IF K>=1 THEN IF K<=5 THEN 130
120 PRINT"О Ш И Б К А":GOTO 100
130 S=S-K
140 PRINT"МНЕ ОСТАЛОСЬ";S
150 IF S<=5 THEN 210
160 K=6-K
170 S=S-K
180 PRINT "Я БЕРУ";K
185 IF S<=5 THEN 210
190 PRINT"ОСТАЛОСЬ";S
200 GOTO 100
210 PRINT"БЕРУ ВСЕ!П О Б Е Д А!"
220 END
```

```
ПРОТОКОЛ РАБОТЫ:
Н А Ч А Л О 30  ПРЕДМЕТОВ
ВАШ ХОД 5
МНЕ ОСТАЛОСЬ 25
Я БЕРУ 1
ОСТАЛОСЬ 24
ВАШ ХОД 5
МНЕ ОСТАЛОСЬ 19
Я БЕРУ 1
ОСТАЛОСЬ 18
ВАШ ХОД 3
МНЕ ОСТАЛОСЬ 15
Я БЕРУ 3
ОСТАЛОСЬ 12
ВАШ ХОД 5
МНЕ ОСТАЛОСЬ 7
Я БЕРУ 1
ОСТАЛОСЬ 6
ВАШ ХОД 5
МНЕ ОСТАЛОСЬ 1
БЕРУ ВСЕ!П О Б Е Д А!
```

Рис. 2.15

Пример 2.I0. Есть отрезки труб длиной A и B метров ($A>B$). Сколько тех и других понадобится, чтобы проложить трубопровод длиной T метров с минимальным количеством сварных швов (трубы разрезать нельзя)?

Протокол работы программы при различных значениях T, A, B приведен на рис. 2.I6, сразу после текста программы. Советуем разобрать самостоятельно алгоритм, по которому работает программа.

```
10  REM ТРУБОПРОВОД ДЛИНОЙ Т МЕТРОВ
20  REM ИЗ А- И В-МЕТРОВЫХ ТРУБ
30  INPUT"ВВЕДИ Т,А,В";T,A,B
40  I=0:TC=T
50  P=INT(T/A)
60  R=PXA
70  R=T-R
80  IF R<>0 THEN 120
90  PRINT A;"-МЕТРОВЫХ:";P
100 PRINT B;"-МЕТРОВЫХ:";I
110 GOTO 160
120 I=I+1
130 T=T-B
140 IF T>0 THEN 50
150 PRINT"ТРУБОПРОВОД ДЛИНОЙ";TC;"М"
160 PRINT"ИЗ";A;"И";B;"-МЕТРОВЫХ ТРУБ"
170 PRINT"  НЕЛЬЗЯ ПОСТРОИТЬ"
180 END


ПРОТОКОЛ РАБОТЫ:


T= 157 A= 7 B= 5
 7 -МЕТРОВЫХ: 21
 5 -МЕТРОВЫХ: 2


T= 18 A= 7 B= 5
ТРУБОПРОВОД ДЛИНОЙ 18 М
ИЗ 7 И 5 -МЕТРОВЫХ ТРУБ
  НЕЛЬЗЯ ПОСТРОИТЬ
```

Рис. 2.I6

Упражнения

I. Даны координаты трех точек $(x_1, y_1), (x_2, y_2), (x_3, y_3)$. Вывести на экран текст "Да", если эти точки лежат на одной прямой, и "Нет" – в противном случае.

2. Даны координаты точки на плоскости: (x, y). Вычислить значение переменной z по следующим правилам:

а) $z = \begin{cases} 1, & \text{если } (x,y) \in \mathcal{D}, \\ 0, & \text{если } (x,y) \notin \mathcal{D}; \end{cases}$ б) $z = \begin{cases} x+y, & \text{если } (x,y) \in \mathcal{D}, \\ x-y, & \text{если } (x,y) \notin \mathcal{D}; \end{cases}$

в) $z = \begin{cases} x^2+y^2, & \text{если } (x,y) \in \mathcal{D}, \\ x^2-y^2, & \text{если } (x,y) \notin \mathcal{D}, \end{cases}$ г) $z = \begin{cases} \sin x, & \text{если } (x,y) \in \mathcal{D}, \\ \cos y, & \text{если } (x,y) \notin \mathcal{D}. \end{cases}$

Варианты задания области $\mathcal{D}$ изображены на рис. 2.17 (заштрихованная часть плоскости).

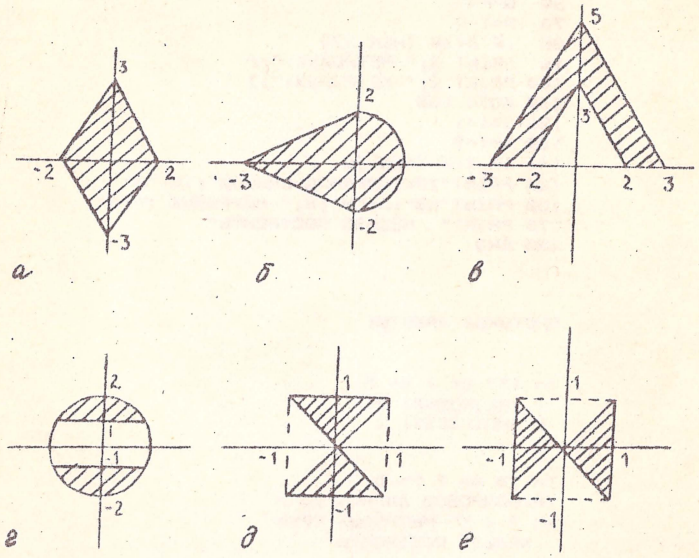


Рис. 2.17

3. Треугольник задан координатами своих вершин: (x_1, y_1), (x_2, y_2), (x_3, y_3). Определить, лежит ли точка с координатами (x, y) внутри треугольника.

4. Заданы координаты трех различных точек на плоскости. Найти ту из них, которая: а) ближе других к началу координат; б) дальше других от начала координат.

3. ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

Циклическими называются алгоритмы, которые содержат многократно повторяемые участки.

В основе циклических алгоритмов лежит базовая алгоритмическая структура ЦИКЛ (ПОВТОРЕНИЕ) в двух ее разновидностях: ЦИКЛ-ПОКА (рис. 3.1,а) и ЦИКЛ-ДО (рис. 3.1,б).

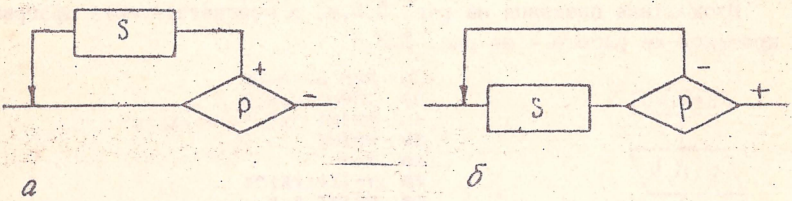


Рис. 3.1

Здесь P - условный оператор; S - повторяющаяся часть алгоритма (тело цикла).

Программные аналоги базовой структуры ЦИКЛ легко реализуются с использованием оператора *IF* . Ниже (рис. 3.2) приведены схемы программ на языке БЕЙСИК для обеих базовых структур. Номера строк программы выбраны произвольно.

```
1Ø IF P THEN 3Ø
2Ø  S  :GOTO 1Ø
3Ø выход
```

ЦИКЛ-ПОКА

а

```
1Ø  S
2Ø IF P THEN 1Ø
3Ø выход
```

ЦИКЛ-ДО

б

Рис. 3.2

Ясно, что в общем случае тело цикла S может занимать несколько строк программы. В некоторых случаях (в зависимости от расположения тела цикла S - на ветви "+" или "-") при записи оператора *IF* условие P заменяется его отрицанием, как и при программировании базовой структуры РАЗВИЛКА, что обусловлено выбором наиболее целесообразного способа программного представления алгоритма.

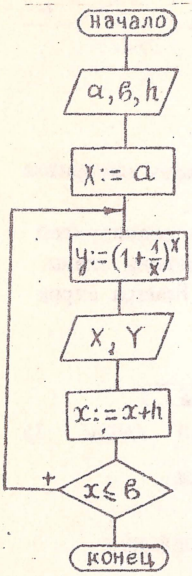
Рассмотрим соответствующие примеры.

Пример 3.1. Составить программу формирования и печати таблицы значений функции

$$y=(1+\frac{1}{x})^x$$

при изменении аргумента x от $a=1$ до $b=101$ с шагом $h=100$.

Блок-схема показана на рис. 3.3,а, а соответствующая программа и протокол ее работы – на рис. 3.3,б.



а

```
10  REM ЦИКЛ-ДО
30  INPUT A,B,H
40  PRINT "X","(1+1/X)^X"
50  PRINT
60  X=A
70  Y=(1+1/X)^X
80  PRINT X,Y
90  X=X+H
100 IF X<=B THEN 70
110 PRINT"КОНЕЦ"
120 END
```

ПРОТОКОЛ РАБОТЫ:

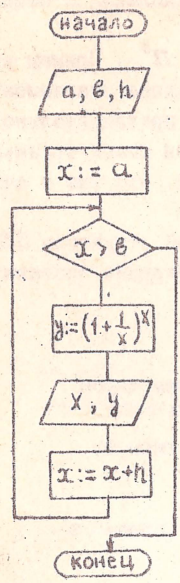
```
X             (1+1/X)^X

 1            2
 101          2.7049459774783
 201          2.7115506254425
 301          2.7137801179485
 401          2.7149001791511
 501          2.7155739265189
 601          2.7160238064967
 701          2.7163455002406
 801          2.7165869629455
 901          2.7167748810307
КОНЕЦ
```

б

Рис. 3.3

Каждое очередное значение вычисляемой функции печатается вместе с соответствующим значением аргумента; оператор в строке 40 выводит заголовок. В результате работы программы на экран дисплея будет выдана таблица (см. рис. 3.3,б).

Этот алгоритм может быть реализован и по схеме ЦИКЛ-ПОКА. Блок-схема и соответствующая программа изображены на рис. 3.4.



а

```
10  REM ЦИКЛ-ПОКА
30  INPUT A,B,H
40  PRINT "X","(1+1/X)^X"
50  PRINT
60  X=A
70  IF X>B THEN 120
80  Y=(1+1/X)^X
90  PRINT X,Y
100 X=X+H
110 GOTO 70
120 PRINT"КОНЕЦ"
130 END
```

ПРОТОКОЛ РАБОТЫ:

X	(1+1/X)^X
1	2
101	2.7049459774783
201	2.7115506254425
301	2.7137801179485
401	2.7149001791511
501	2.7155739265189
601	2.7160238064967
701	2.7163455002406
801	2.7165869629455
901	2.7167748810307

КОНЕЦ

б

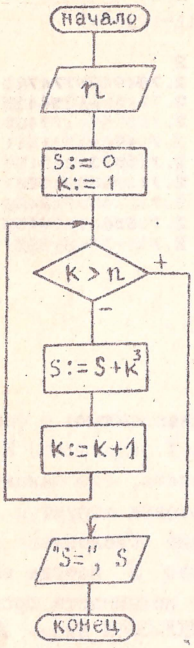
Рис. 3.4

Приведенные на рис. 3.3 и 3.4 программы равносильны в том смысле, что они дадут в данном случае одинаковый результат. И тем не менее программист должен ясно себе представлять, что одинаковые по смыслу программы, составленные на основе базовых структур ЦИКЛ-ДО и ЦИКЛ-ПОКА, все-таки различны. Различие обусловлено самой природой этих структур: в структуре ЦИКЛ-ДО тело S всегда выполняется по крайней мере один раз, независимо от истинности проверяемого условия P (см. рис. 3.I); в структуре ЦИКЛ-ПОКА блок S может не выполняться ни разу. В частности, если в рассмотренном примере верхняя граница параметра цикла X заранее не известна и определяется в результате некоторых предварительных вычислений, то может оказаться, что первая программа (ЦИКЛ-ДО) в этой ситуации даст неверный результат. Действительно, если окажется, что $B<A$, то вторая программа (ЦИКЛ-ПОКА) сразу же завершит работу, в то время как первая вычислит и выведет на экран одну строку, после чего остановится. Подобные ситуации требуют особого внимания при составлении программ.

Пример 3.2. Составить БЕЙСИК-программу вычисления и печати суммы кубов натуральных чисел от I до n .

Требуется найти $S = I + 2^3 + 3^3 + \ldots + n^3$. Примем вначале $S = 0$ и будем добавлять к искомой сумме очередное слагаемое K^3, начиная с $K = I$. Перед операцией суммирования предварительно будем проверять выполнение условия $K \leqslant n$. Только при такой организации алгоритма результат будет верным, если окажется, что уже в самом начале $n \leqslant 0$.

Блок-схема алгоритма, основанная на базовой структуре ЦИКЛ-ПОКА, изображена на рис. 3.5,а, пользуясь которой, нетрудно составить и программу (рис. 3.5,б).



а

```
1Ø REM СУММА КУБОВ
2Ø INPUT N:PRINT"N=";N
3Ø S=Ø:K=1
4Ø IF K>N THEN 8Ø
5Ø S=S+K^3
6Ø K=K+1
7Ø GOTO 4Ø
8Ø PRINT "СУММА=";S
9Ø END

ПРОТОКОЛ РАБОТЫ:

N= 8
СУММА= 1296
```

б

Рис. 3.5

Аналогично составляются программы вычисления суммы n членов любой последовательности, заданной общим членом $a_k = f(K)$

Пример 3.3. Составить программу вычисления значения функции $y = a^n$, где n – целое неотрицательное число.

Алгоритм сводится к многократному повторению операции умножения $1 \cdot a \cdot a \cdot a \ldots a$. Поэтому вначале полагаем $y = I$, затем при каждом

обороте цикла будем последовательно умножать y на a, принимать полученный результат за новое значение y и т.д., пока операция умножения не выполнится требуемое число раз. Блок-схема этого алгоритма подобна схеме предыдущего примера, поэтому приводим только программу:

```
10 REM СТЕПЕНЬ
20 DATA 5,0,2,7
30 READ A,N
40 Y=1:K=1
50 IF K>N THEN 80
60 Y=Y*A
70 K=K+1:GOTO 50
80 PRINT A;"В";N"-Й СТЕПЕНИ=";Y
90 GOTO 30

ПРОТОКОЛ РАБОТЫ:

 5 В 0 -Й СТЕПЕНИ= 1
 2 В 7 -Й СТЕПЕНИ= 128
```

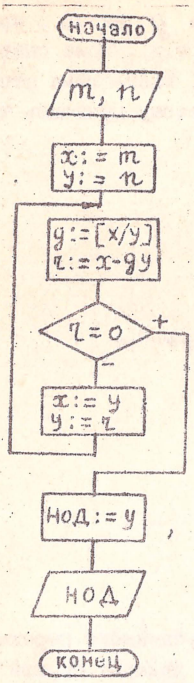
Программа заканчивается оператором безусловного перехода *GOTO 30* и благодаря этому будет выполнена дважды: первый раз при $A = 5$, $N = 0$; второй раз при $A = 2$, $N = 7$. Обратите внимание на то, как составлено тело оператора *PRINT* и в каком виде печатаются результаты.

Пример 3.4. Даны два целых положительных числа m и n. Составить программу вычисления их наибольшего общего делителя, используя алгоритм Евклида.

Блок-схема показана на рис. 3.6,а, а соответствующая программа – на рис. 3.6,б.

Чтобы сохранить для последующего вывода исходные числовые значения M и N, в начале программы (строка 40) эти величины присваиваются рабочим переменным X и Y. Оператор $Q = INT(X/Y)$ находит целую часть частного X/Y и присваивает результат переменной Q; оператор $R = X - Q*Y$ находит остаток от деления X на Y*.

* Более подробно этот алгоритм уже обсуждался в выпуске: Основы алгоритмизации: Учеб. пособие для учителей, читающих "Основы информатики и вычислительной техники в 9 классах средней школы" /Сост. В.С.Макогон. – Одесса: ОГУ, 1985.



а

```
10  REM НАИБОЛЬШИЙ ОБЩИЙ ДЕЛИТЕЛЬ
20  REM АЛГОРИТМ ЕВКЛИДА
30  INPUT M,N
40  X=M:Y=N
50  Q=INT(X/Y)
60  R=X-QXY
70  IF R=0 THEN 100
80  X=Y:Y=R
90  GOTO 50
100 H=Y
110 PRINT "НОД(";M;",";"N")=";H
120 END
```

ПРОТОКОЛ РАБОТЫ:

НОД(14 , 35)= 7

б

Рис. 3.6

Оператор печати выдает результат в виде $НОД(M,N)=Y$, где M и N – исходные целые числа, а Y – найденный наибольший общий делитель этих чисел. Например, если $M=14$ и $N=21$, то результат будет выведен на экран в форме:

$$НОД(14,21)=7$$

В приведенном примере количество повторений цикла заранее не определено. Такие циклические алгоритмы, в которых заранее не известно число шагов, необходимых для решения задачи, и проверка окончания вычислений происходит не по счетчику повторений, а по достижении нужной точности вычислений, называются, как известно, итерационными.

Пример 3.5. Рассмотрим задачу извлечения квадратного корня

$$y=\sqrt{a}$$

без использования стандартной функции БЕЙСИКа.

Для решения задачи воспользуемся итерационной формулой Ньютона:

$$y_{n+1}=\frac{1}{2}\left(y_n+\frac{a}{y_n}\right)=y_n+\frac{1}{2}\left(\frac{a}{y_n}-y_n\right),\ n=0,1,2,\ldots \qquad (3.1)$$

Эта формула позволяет вычислять очередной член последовательности $\{y_n\}$ через уже вычисленный предыдущий член, начиная с некоторого начального приближения y_0, которое задается произвольно. В математике доказано, что такая последовательность стремится (сходится) к точному значению $\sqrt{a}$. Поскольку мы не можем позволить себе строить бесконечную последовательность приближений y_0, y_1, y_2, ... y_n, ..., то процесс обрывается при некотором n таким образом, чтобы погрешность вычисления $y=\sqrt{a}$ не превосходила некоторой заданной величины $\varepsilon>0$. Для этого достаточно сравнить два последовательных приближения y_{n+1} и y_n. Если

$$|y_{n+1}-y_n|<\varepsilon, \qquad (3.2)$$

то вычисления следует прекратить и полученное приближение y_{n+1} принять за значение $\sqrt{a}$.

Из формулы (3.1) следует, что

$$|y_{n+1}-y_n|=\frac{1}{2}\left|\frac{a}{y_n}-y_n\right|$$

Введем обозначение: $v=\frac{1}{2}\left(\frac{a}{y_n}-y_n\right)$. Тогда условие (3.2) примет вид $|v|<\varepsilon$.

При программировании индексы в итерационной формуле не нужны, так как в операторе $y:=y+v$ слева от знака присваивания стоит "новое", $(n+1)$-е значение, а справа – "старое", n-е значение переменной y, что и требуется согласно алгоритму.

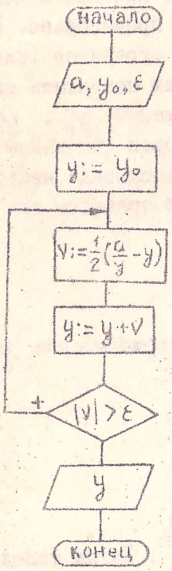
Блок-схема алгоритма изображена на рис. 3.7,а, соответствующая БЕЙСИК-программа – на рис. 3.7,б.

По итерационной схеме вычисляют многие элементарные функции, используя их разложения в степенные ряды. Наиболее употребительные степенные ряды для функций e^x, $\sin x$, $\cos x$, которые сходятся при любом значении x:

$$e^x=1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots+\frac{x^n}{n!}+\ldots$$

$$\cos X = 1 - \frac{X^2}{2!} + \frac{X^4}{4!} - \ldots + (-1)^n \frac{X^{2n}}{(2n)!} + \ldots$$

$$\sin X = X - \frac{X^3}{3!} + \frac{X^5}{5!} - \ldots + (-1)^n \frac{X^{2n+1}}{(2n+1)!} + \ldots$$



а

```
10 REM КВАДРАТНЫЙ КОРЕНЬ
20 REM ПО АЛГОРИТМУ НЬЮТОНА
30 INPUT A,Y0,E
40 Y=Y0
50 V=.5*(A/Y-Y)
60 Y=Y+V
70 IF ABS(V)>E THEN 50
80 PRINT "КОРЕНЬ ИЗ";A;"=";Y
90 END

ПРОТОКОЛ РАБОТЫ:

КОРЕНЬ ИЗ 9.05 = 3.0083217927952
```

б

Рис. 3.7

Степенные ряды являются важным аппаратом для табулирования элементарных функций, так как их применение позволяет свести задачу вычисления значений функции с заданной точностью к задаче вычисления многочлена.

Приведем здесь для справок разложения в степенные ряды еще нескольких популярных функций:

$$\operatorname{arctg} X = X - \frac{X^3}{3} + \frac{X^5}{5} - \ldots + (-1)^n \frac{X^{2n+1}}{2n+1} + \ldots$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \dots + (-1)^{n-1}\frac{x^n}{n} + \dots \quad (|x|<1)$$

$$\ln\frac{1+x}{1-x} = 2\left(x + \frac{x^3}{3} + \frac{x^5}{5} + \dots + \frac{x^{2n+1}}{2n+1} + \dots\right)$$

При вычислении элементарных функций с помощью степенных рядов удобно пользоваться рекуррентными соотношениями, которые позволяют вычислять очередной член ряда не непосредственно (такое решение было бы слишком примитивным), а через уже вычисленные предыдущие члены.

Для приведенных рядов рекуррентные соотношения могут быть выведены непосредственно. Достаточно взять отношение двух соседних членов ряда. Рассмотрим, например, степенный ряд для функции e^x, общий член которого имеет вид $U_n = X^n/n!$ Вычисляя отношение последующего члена к предыдущему, получаем:

$$\frac{U_{n+1}}{U_n} = \frac{X^{n+1}\,n!}{(n+1)!\,X^n} = \frac{x}{n+1}, \quad n = 0, 1, 2, \dots$$

Таким образом, для двух соседних членов ряда получаем рекуррентное соотношение

$$U_{n+1} = U_n \cdot x/(n+1), \quad n = 0, 1, 2, \dots,$$

которое удобно для последовательного вычисления членов ряда (при этом мы избавлены от необходимости многократного вычисления степеней и факториалов).

При вычислениях с помощью степенных рядов мы ограничиваемся конечным числом членов, заменяя сумму ряда его частичной суммой. Возникает практический вопрос: сколько членов ряда нужно сохранить, чтобы получающаяся погрешность не превышала заданной?

Для знакопеременных рядов погрешность легко оценивается: при замене суммы такого ряда его частичной суммой допускается погрешность, не превосходящая по модулю первого из отброшенных членов.

Для рядов с положительными членами оценка погрешности более сложная и здесь не рассматривается.

Пример 3.6. Составим программу вычисления функции $y = \sin x$ с заданной точностью $\varepsilon > 0$, пользуясь ее разложением в степенной ряд:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots \qquad (3.3)$$

Теоретически этот ряд пригоден для любых значений X, однако в действительности он практически бесполезен для больших значений X из-за очень больших ошибок округления. Будем предполагать, что $X \leq 2\pi$, в противном случае надо пользоваться сначала формулами приведения, а затем использовать для вычисления $\sin X$ степенной ряд (3.3).

Введем обозначения:

$$U_0 = X,\ U_1 = -\frac{X^3}{3!},\ U_2 = \frac{X^5}{5!},\ \dots,\ U_n = (-1)^n \frac{X^{2n+1}}{(2n+1)!}$$

Тогда исходный ряд (3.3) можно переписать так:

$$S \equiv \sin X = U_0 + U_1 + U_2 + \dots + U_n + U_{n+1} + \dots \qquad (3.4)$$

Частичные суммы последнего ряда

$$S_0 = U_0,\ S_1 = S_0 + U_1,\ \dots,\ S_{n+1} = S_n + U_{n+1}$$

являются последовательными приближениями для искомого значения функции $S = \sin X$.

Формула (3.4) может быть запрограммирована в виде циклического итерационного процесса, который описывается рекуррентными соотношениями между членами ряда U_n и частичными суммами S_n:

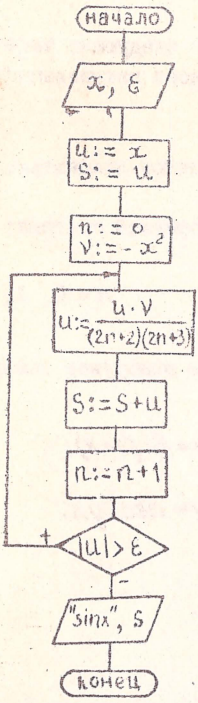
$$U_{n+1} = U_n \frac{-X^2}{(2n+2)(2n+3)},$$

$$S_{n+1} = S_n + U_{n+1}, \qquad n = 0, 1, 2, \dots$$

Из этих соотношений видно, что величины U_{n+1}, S_{n+1} могут быть получены на основе величин U_n, S_n, n, вычисленных в предыдущем исполнении цикла. Начальные значения этих величин $n = 0$, $S_0 = X$, $U_0 = X$ определяются на основе формулы (3.3). Так как разность между соседними значениями функции $S_{n+1} - S_n = U_{n+1}$, то момент окончания вычислений наступит при

$$|U_{n+1}| \leq \varepsilon.$$

Составим соответствующую блок-схему описанного алгоритма (рис. 3.8,а) и БЕЙСИК-программу (рис. 3.8,б) (при составлении блок-схемы и программы мы отказались от использования индексов по соображениям, приведенным в предыдущем примере).



а

```
1Ø  REM SINX
2Ø  INPUT X,E
3Ø  PRINT "X="X;"ТОЧНОСТЬ="E
4Ø  U=X:S=U
5Ø  N=Ø:V=-(X*X)
6Ø  Z=(2*N+2)*(2*N+3)
7Ø  U=U*V/Z
8Ø  S=S+U
9Ø  N=N+1
1ØØ IF ABS(U)>E THEN 6Ø
11Ø PRINT "SIN"X"=";S
12Ø END
```

ПРОТОКОЛ РАБОТЫ:

```
X= 1.52 ТОЧНОСТЬ= 1E-04
SIN 1.52 = .9987101Ø725387
```

б

Рис. 3.8

В программе предполагается, что аргумент X задан в радианах. Если же он задан в градусах, то в начале программы достаточно вставить оператор перевода градусной меры в радианную, например:

$$25X=X*P1/18Ø$$

В этом операторе X справа от знака операции присваивания выражен в градусах, а слева – уже в радианах.

В программе введены промежуточные переменные: а) чтобы не вычислять каждый раз в теле цикла – X^2, определяем его однажды за

пределами цикла и присваиваем полученное значение переменной V ;
б) переменной Z обозначено числовое значение знаменателя, вычисляемое каждый раз с новым значением переменной N .

Упражнения

Составить БЕЙСИК-программы для решения следующих задач.

I. Найти сумму квадратов всех нечетных чисел натурального ряда от I до n .

2. Вычислить $y = n!$

3. Вычислить $y = \sqrt[p]{a}$, $a > 0$, p – целое положительное число.

Указание. Воспользоваться итерационной формулой Ньютона:

$$y_{n+1} = y_n + \frac{1}{p}\left(\frac{a}{y_n^{p-1}} - y_n\right), \qquad n = 0, 1, 2, \ldots$$

4. Вычислить с заданной точностью ε , не пользуясь стандартными функциями языка БЕЙСИК:

а) $y = \cos x$; б) $y = \ln(1+x)$;

в) $y = \frac{e^x + e^{-x}}{2}$; г) $y = \operatorname{arc}\operatorname{tg} x$.

5. Табулированием трехчленов:

$$4X^2 - 162X + 1681$$
$$9X^2 - 249X + 1763$$

найти последовательности простых чисел; X изменяется от I до 40 с шагом I.

Результат выдать на экран дисплея и снабдить заголовком "ПРОСТЫЕ ЧИСЛА".

6. Вычислить сумму n членов последовательности с общим членом:

а) $a_k = \frac{2k}{k^2+1}$; б) $a_k = \frac{\sqrt{k}}{3k+1}$.

7. Что будет выдано на экран дисплея в результате работы такой программы:

```
10 N=35
20 I=0:K=1
30 PRINT K;
40 K=K+I
50 I=K-1
60 IF K<=N THEN 30
70 END
```

8. Вычислить и напечатать последовательность, заданную общим членом следующего вида:

а) $x_n = \frac{n^2 + 2n - 1}{2n^2 - 3n + 1}$; б) $x_n = \frac{1 \cdot 3 \cdot 5 \ldots (2n-1)}{2 \cdot 4 \cdot 6 \ldots 2n}$;

в) $x_n = \left(1 + \frac{1}{1!}\right)\left(1 + \frac{1}{2!}\right) \ldots \left(1 + \frac{1}{n!}\right)$;

г) $x_n = \sum_{k=1}^{n} (-1)^k \frac{k!}{(2k)!}$.

Вычисления продолжать, пока $|x_{n+1} - x_n| > 0{,}001$.

9. Найти и напечатать число членов и приближенное значение суммы S:

а) $S = \left(1 + \frac{1}{2}\right) - \left(\frac{1}{3} + \frac{1}{4}\right) + \left(\frac{1}{5} + \frac{1}{6}\right) - \left(\frac{1}{7} + \frac{1}{8}\right) + \ldots$

б) $S = 1 - \left(\frac{1}{2} + \frac{1}{3}\right) + \left(\frac{1}{4} + \frac{1}{5} + \frac{1}{6}\right) - \left(\frac{1}{7} + \frac{1}{8} + \frac{1}{9} + \frac{1}{10}\right) + \ldots$

Процесс суммирования продолжать до тех пор, пока члены суммы по модулю больше 0.0001.

10. Найти и напечатать число членов и приближенное значение суммы

$$S = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{8} + \frac{1}{13} + \ldots,$$

в которой знаменатель каждого слагаемого равен сумме знаменателей двух предыдущих членов. Вычисления закончить, когда очередное слагаемое станет меньше 0.0005.

11. Последовательность Фибоначчи (см. упражнение 7) задается соотношениями вида

$$f_n = f_{n-1} + f_{n-2}, \quad n = 2, 3, \ldots; \quad f_0 = 1, \; f_1 = 1.$$

Требуется: а) найти число членов последовательности, не превосходящих 1000;

б) найти сумму членов последовательности, не превосходящих заданного числа N ;

в) напечатать порядковые номера членов последовательности, удовлетворяющих условию $K < f_n < L$, где K и L – заданные числа;

г) найти максимальное число членов последовательности, сумма которых не превосходит заданной величины Q .

3.I. Оператор цикла

Циклические алгоритмы могут быть описаны следующей схемой:

I) установка начальных значений, участвующих в цикле переменных;

2) задание закона изменения переменной, являющейся параметром цикла;

3) проверка условия окончания цикла: выход из цикла при выполнении этого условия или переход к следующему шагу в противном случае;

4) реализация повторяющейся части алгоритма (тела цикла);

5) возврат к шагу 2.

Эта схема на БЕЙСИКе успешно реализуется с помощью нескольких операторов присваивания и оператора *IF* , что и было наглядно проиллюстрировано в предыдущих примерах.

Большинство встречающихся в программах циклов удовлетворяет таким условиям:

число повторений цикла конечно и известно до начала его выполнения;

закон изменения параметра цикла – арифметическая прогрессия с заданной разностью и начальным членом.

Этим условиям удовлетворяют, в частности, практически все циклические алгоритмы обработки массивов данных.

Для программирования таких циклов в БЕЙСИКе имеется специальная конструкция, называемая оператором цикла, которая обеспечивает их запись в более компактной и удобной для восприятия форме. Она состоит из двух специальных операторов: оператора начала цикла (или, коротко, оператора *FOR*) и оператора конца цикла (оператора *NEXT*). Операторы *FOR* и *NEXT* всегда используются совместно.

Оператор *FOR* (его еще называют заголовком цикла) имеет вид

```
FOR I=A TO B STEP C
```

или

```
FOR I=A TO B
```

Здесь *FOR*, *TO*, *STEP* - служебные слова, которые можно трактовать соответственно как ДЛЯ, ДО, ШАГ;

I - переменная (параметр) цикла; в качестве параметра цикла должна использоваться простая (неиндексированная) переменная;

A, *B*, *C* - соответственно начальное значение, конечное значение и величина шага изменения параметра цикла. Они могут быть любыми выражениями. Когда шаг цикла равен единице, используется вторая форма записи оператора (без служебного слова *STEP*).

Примеры.

```
10 FOR K=1 TO 100
20 FOR X=X1 TO X2 STEP H
40 FOR M=10 TO 1 STEP-1
```

Последний пример напоминает, что шаг цикла может быть и отрицательным.

Оператор *NEXT* имеет вид

NEXT <переменная цикла>

Примеры.
```
50 NEXT K
60 NEXT M
```

В паре операторов *FOR* и *NEXT* должна использоваться одна и та же переменная цикла. Общая структура оператора цикла такова:

```
FOR I=A TO B [STEP H]
        <тело цикла>
NEXT I
```

При выполнении оператора цикла происходит следующее.

1. Параметру цикла *I* присваивается значение, равное начальному значению *A*.

2. Значение параметра цикла сравнивается с его конечным значением *B*. Если оно больше (меньше в случае отрицательного значения шага *H*) конечного значения параметра цикла, то цикл завершает работу и управление передается оператору, следующему за оператором *NEXT*. В противном случае выполняется шаг 3.

3. Последовательно выполняются операторы, расположенные между *FOR* и *NEXT* (тело цикла).

4. По достижении оператора *NEXT* происходит изменение значения параметра цикла *I* на величину шага *H* и снова выполняется шаг 2.

Рассмотрим несколько примеров.

Пример 3.7. Используя оператор цикла, составить программу вычисления суммы $S = I + 2^3 + 3^3 + \ldots + N^3$. Сравнить полученную программу с решением примера 3.2 (рис. 3.9).

```
10 REM СУММА КУБОВ
20 INPUT N
30 PRINT "N="N
40 S=0
50 FOR K=1 TO N
60 S=S+K^3
70 NEXT K
80 PRINT "СУММА=",S
90 END
ПРОТОКОЛ РАБОТЫ:

СУММА=           441
```

Рис. 3.9

В начале работы оператора цикла (для наглядности он на рис. 3.9 заключен в рамку) переменная *K* получает значение, равное единице. После каждого выполнения оператора 60 (тело цикла), где происходит добавление к предыдущей сумме куба очередного натурального числа, значение параметра цикла *K* увеличивается на единицу (такова величина шага по умолчанию) и сравнивается со значением переменной *N*. Цикл повторяется до тех пор, пока выполнено соотношение $K \leqslant N$, после чего печатается значение накопленной суммы.

Пример 3.8. Для заданного *X* вычислить произведение

$$P = \prod_{K=1}^{50} \left(\frac{1-KX}{1+KX} \right).$$

Соответствующая программа может иметь вид:

```
10 REM ПРОИЗВЕДЕНИЕ
20 INPUT X:PRINT "X=";X
30 P=1
40 FOR K=1 TO 50
50   P=P*(1-K*X)/(1+K*X)
60 NEXT K
70 PRINT "ПРОЗВЕДЕНИЕ=",P
80 END
ПРОТОКОЛ РАБОТЫ:
X= 999
ПРОЗВЕДЕНИЕ=    .991033026372222
```

4. ОБРАБОТКА ТАБЛИЦ (МАССИВОВ) В БЕЙСИК-ПРОГРАММЕ

Наиболее полно преимущества оператора цикла проявляются при обработке массивов.

4.1. Оператор описания массивов

В разд. I.5 подробно рассмотрена возможность использования наряду с простыми переменными переменных с индексами – элементов одномерных и двумерных массивов. Все массивы, используемые в программе, должны быть обязательно предварительно описаны. Описание массива содержит информацию о количестве элементов и его размерности. Такая информация необходима интерпретатору БЕЙСИКа для резервирования определенной области оперативной памяти и выделения необходимого количества мест для элементов массива.

Для описания массива используется оператор *DIM*, который в программе записывается следующим образом: вслед за словом *DIM* помещается (лучше через пробел) имя массива, а за ним в круглых скобках – верхние границы индексов (для двумерных массивов указываются две границы, разделенные запятыми). Одним оператором *DIM* можно описать несколько массивов, разделяя их описания запятыми. Например, оператор

```
10 DIM A(5), X(2,3)
```

описывает два массива: одномерный массив A из шести элементов и двумерный массив X с тремя строками и четырьмя столбцами, которые мы изобразим схематически в виде отдельных ячеек-элементов массивов.

Массив A :

A(0)	A(1)	A(2)	A(3)	A(4)	A(5)

Массив X :

X(0,0)	X(0,1)	X(0,2)	X(0,3)
X(1,0)	X(1,1)	X(1,2)	X(1,3)
X(2,0)	X(2,1)	X(2,2)	X(2,3)

(Естественно, что память ЭВМ линейна и в ней никаких прямоугольных таблиц, подобно приведенной выше, не наблюдается.) Часто в алгорит-

мах индексы элементов массивов изменяются не от нуля, а от единицы и до максимально возможного значения, указанного при описании соответствующего массива. Не следует, однако, забывать о том, что для "нулевых" элементов отводится место в памяти. В частности, это обстоятельство надо иметь в виду при вводе массивов в память ЭВМ.

Оператор *DIM* в программе должен быть расположен до использования описываемых им массивов. Всем элементам, описанных операторы *DIM* массивов, присваиваются нулевые значения. Индексы в индексированных переменных и операторе *DIM* могут быть заданы с помощью арифметических выражений. Значение такого выражения должно быть неотрицательным числом. За значение индекса принимается целая часть (без округления) значения вычисленного арифметического выражения. Например, оператор

```
10 DIM A(I*J,K)
```

при текущих значениях переменных $I = 2.8$, $J = 3$, $K = 7.9$ определяет двумерный массив A из восьми строк, семи столбцов.

Повторное определение массивов, а также массива, имеющего имя уже определенной простой переменной, не допускается.

В дальнейшем, говоря, что задан массив $A(N)$, будем понимать, что он описан в программе как

```
DIM A(N)
```

и порождает в ней элементы $A(0)$, $A(1)$, ..., $A(N)$

4.2. Ввод и вывод массивов в БЕЙСИК-программе

Присвоение числовых значений элементам массива может быть осуществлено, как и для неиндексированных (простых) переменных, в операторе присваивания и операторах ввода данных, например:

```
10 X(7)=5.3 : Z(K,3)=9.37E-3
20 INPUT X(1), X(2), X(3), X(4)
30 DATA -5.3, 0, 8.5, 194, -98
40 READ Z(1,1), Z(1,2), Z(1,3)
```

При необходимости внесения в программу больших объемов информации этот способ ввода массивов неудобен. Наиболее просто можно осу-

ществить ввод массива в отдельных реализациях БЕЙСИКа, указав в соответствующем операторе ввода (*INPUT*, *READ*) только имя массива и вслед за ним - открывающую и закрывающую скобки. Например, последовательность операторов в БЕЙСИКе для "Электроники ДЗ-28"

```
10 DIM A(3)
20 DATA 4,2,5,3.7
30 READ A( )
```

присвоит: $A(0)=4; A(1)=2; A(2)=5; A(3)=3.7$ Не следует забывать, что нумерация элементов массива начинается с нуля.

Оператор непосредственного ввода данных в режиме диалога с ЭВМ также часто используется для ввода массивов:

```
10 DIM A(3)
20 INPUT A( )
```

В момент выполнения последнего оператора *INPUT* машина останавливается и выводит на экран символ "?", после чего программист должен ввести с клавиатуры 4 (четыре!) числовые значения, разделенные запятыми. Они становятся значениями элементов массива $A(0)$, $A(1)$, $A(2)$, $A(3)$. Если набранное количество чисел не соответствует длине массива, то на экране появляется сообщение об ошибке.

При вводе большого массива ошибиться нетрудно. Поэтому рекомендуется использовать следующий шаблон, комбинирующий оператор вывода - подсказку о том, какой элемент массива следует вводить, - с оператором ввода *INPUT* :

```
10 INPUT N
20 DIM A(N)
30 FOR I=1 TO N
40  PRINT "A("I")="
50  INPUT A(I)
60 NEXT I
```

При выполнении этого фрагмента программы на экран последовательно вводятся строки

```
A(1) =
?
A(2) =
?
...
```

до тех пор, пока все N элементов массива A не будут введены. Разумеется, после каждой подсказки и следующего за ней символа "?" машина останавливается и надо вводить с клавиатуры очередной элемент массива.

При выводе массивов на печать или экран дисплея следует помнить, что если в теле цикла оператор *PRINT* завершается символами "," или ";", то выводимые элементы располагаются построчно, в зонном или плотном форматах соответственно; в противном случае в каждой строке будет выведен один элемент массива.

Проиллюстрируем сказанное следующими примерами.

Пример 4.1. Дан массив $A(N)$. Вычислить и выдать на экран среднеарифметическое всех его положительных компонент.

Блок-схема и программа показаны на рис. 4.1. Для вычисления суммы строго положительных элементов массива заводим переменную S, а для подсчета их количества - переменную K. Обе переменные за пределами цикла полагаем равными нулю. В последующем, если очередной элемент массива $A(I)>0$, то он прибавляется к текущему значению переменной S, а счетчик K увеличивается на единицу.

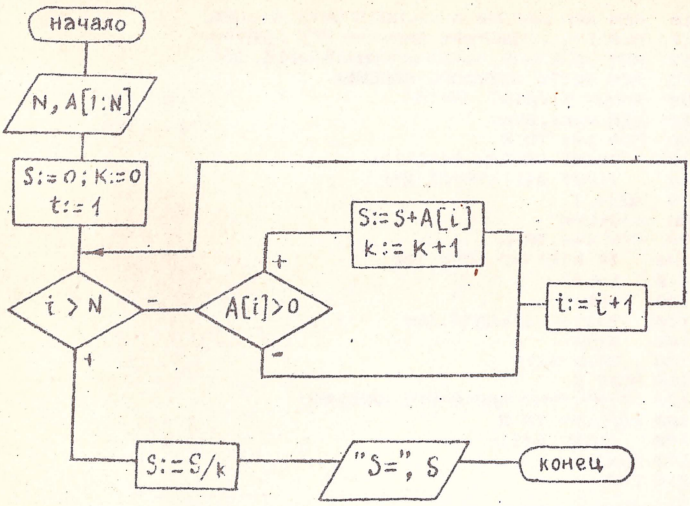
Пример 4.2. По заданному массиву $A(N)$ построить массив $B(N)$, содержащий в начале все неотрицательные, а затем - все отрицательные элементы массива.

Будем заполнять массив B с двух концов: положение очередного неотрицательного элемента в массиве B определяется индексом K, который возрастает $(K=K+1)$, начиная с единицы; положение очередного отрицательного элемента в массиве B определяется индексом J, который последовательно уменьшается $(J=N+1-L)$, начиная с $J=N$.

Текст программы и протокол ее работы показаны на рис. 4.2.

Пример 4.3. Ненулевые элементы массива $A(N)$ переписать с сохранением их порядка в начало массива, а нулевые - в конец его. Новый массив не заводить.

Программа на языке БЕЙСИК для решения поставленной задачи и протокол ее выполнения изображены на рис. 4.3. Первый цикл программы обеспечивает ввод очередного элемента массива и анализ его знака. Положительные компоненты массива A "подтягиваются" к началу массива, нулевые - пропускаются. После того как весь массив просмотрен до конца, в его "хвост" дописывается столько нулей, сколько их было обнаружено в исходном массиве.



а

```
1Ø  REM СРЕДНЕЕ АРИФМЕТИЧЕСКОЕ
2Ø  INPUT N:PRINT"N="N
3Ø  DIM A(N)
4Ø  FOR I=1 TO N
5Ø    PRINT "ВВЕДИ A("I")";
6Ø    INPUT A(I):PRINT A(I)
7Ø  NEXT I
8Ø  S=Ø:K=Ø
9Ø  FOR I=1 TO N
1ØØ   IF A(I)<=Ø THEN 13Ø
11Ø   S=S+A(I)
12Ø   K=K+1
13Ø NEXT I
14Ø IF K=Ø THEN 19Ø
15Ø S=S/K
16Ø PRINT "СРЕДНЕЕ АРИФМЕТИЧЕСКОЕ";K;
17Ø PRINT "СЛАГАЕМЫХ=";S
18Ø GOTO 2ØØ
19Ø PRINT "НЕТ A(I)>Ø"
2ØØ END
```

ПРОТОКОЛ РАБОТЫ:

```
N= 4
ВВЕДИ A( 1 ) 1Ø
ВВЕДИ A( 2 ) 2Ø
ВВЕДИ A( 3 )-5Ø
ВВЕДИ A( 4 ) 4Ø
СРЕДНЕЕ АРИФМЕТИЧЕСКОЕ 3 СЛАГАЕМЫХ= 2Ø.
```

б

Рис. 4.I

```
10  REM ДАН МАССИВ A(N).ПОСТРОИТЬ МАССИВ
11  REM B(N),СОДЕРЖАЩ.ВНАЧАЛЕ ВСЕ НЕОТРИ-
12  REM ЦАТЕЛЬНЫЕ,ЗАТЕМ-ОТРИЦАТЕЛЬНЫЕ ЭЛЕ-
13  REM МЕНТЫ ИСХОДОГО МАССИВА
20  INPUT N:PRINT "N="N
30  DIM A(N),B(N)
40  FOR I=1 TO N
50    PRINT "ВВЕДИ A("I")";
60    INPUT A(I):PRINT A(I)
70  NEXT I
80  K=0:L=0
90  FOR I=1 TO N
100   IF A(I)>=0 THEN 140
110   L=L+1
120   J=N+1-L
130   B(J)=A(I):GOTO 160
140   K=K+1
150   B(K)=A(I)
160 NEXT I
170 PRINT "УПОРЯДОЧЕННЫЙ МАССИВ:"
180 FOR I=1 TO N
190   PRINT B(I);
200 NEXT I
210 END
```

ПРОТОКОЛ РАБОТЫ:

```
N= 7
ВВЕДИ A( 1 )-99
ВВЕДИ A( 2 ) 77
ВВЕДИ A( 3 ) 0
ВВЕДИ A( 4 )-88
ВВЕДИ A( 5 ) 77
ВВЕДИ A( 6 )-66
ВВЕДИ A( 7 ) 55
УПОРЯДОЧЕННЫЙ МАССИВ:
 77  0  77  55 -66 -88 -99
```

Рис. 4.2

```
10  REM НЕНУЛЕВЫЕ ЭЛЕМЕНТЫ МАССИВА ПЕРЕ-
20  REM ПИСАТЬ С СОХРАНЕНИЕМ ПОРЯДКА В
30  REM ЕГО НАЧАЛО,А НУЛЕВЫЕ-В КОНЕЦ
40  DATA 0,8,0,97,-13,0,34,0,0,99
50  INPUT "ВВЕДИ N";N :PRINT "N="N
60  DIM A(N)
70  J=1
80  FOR I=1 TO N
90    READ A(I)
100   IF A(I)<>0 THEN A(J)=A(I):J=J+1
110 NEXT I
120 FOR K=J TO N
130   A(K)=0
140 NEXT K
150 FOR I=1 TO N
160   PRINT A(I);
170 NEXT I
180 END
```

ПРОТОКОЛ РАБОТЫ:

```
N= 10
 8  97 -13  34  99  0  0  0  0  0
```

Рис. 4.3

Упражнения

1. Дан массив A (50). Найти произведение положительных элементов.

2. Даны массивы $A(N)$ и $B(N)$ ($N \leq 100$). Найти значение индекса K такое, что $(A(K)+B(K))/2$ максимально.

3. Дан массив X (100), компоненты которого упорядочены по возрастанию и число Q такое, что $X_1 < Q < B_{100}$. Найти K, удовлетворяющее соотношению $B_K \leq Q < B_{K+1}$.

4. Из компонент массивов $A(N)$ и $B(N)$ построить массив $C=(A_1, B_1, A_2, B_2, \ldots A_N, B_N)$.

5. Какая задача решается при выполнении следующего фрагмента программы:

```
10 P=0:N=0
20 FOR I=1 TO 20
30   IF A(I)<=0 THEN 50
40   P=P+1:GOTO 70
50   IF A(I)=0 THEN 70
60   N=N+1
70 NEXT I
80 PRINT P,N
90 END
```

6. Дан массив $A(N)$ и число z. Вычислить сумму $S=\sum_{i=1}^{n} A_i z^i$ без использования операции возведения в степень.

7. Разделить заданный массив X (100) на два массива:

а) массив положительных и массив отрицательных компонент;

б) массив четных и массив нечетных компонент.

5. СЛОЖНЫЕ ЦИКЛЫ

Сложными традиционно называют циклы, содержащие в себе разветвления и другие вложенные циклы. При реализации вложенных циклов необходимо следить за тем, чтобы параметры внутреннего (вложенного) цикла восстанавливались в первоначальном виде при каждом выполнении внешнего (окаймляющего) цикла).

Пример 5.I. Составить программу печати таблицы Пифагора (рис. 5.I).

```
10  REM ПИФАГОР
20  INPUT N
30  PRINT "      Т А Б Л И Ц А   П И Ф А Г О Р А"
40  PRINT:PRINT
50  FOR I=1 TO N
60   FOR J=1 TO N
70    PRINT TAB(4*J)I*J;
80   NEXT J
90  PRINT
100 NEXT I
110 END
```

Результаты работы программы при N = 9:

Т А Б Л И Ц А П И Ф А Г О Р А

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

Рис. 5.I

Приведенная на рис. 5.I программа достаточно проста, хотя и содержит сложный цикл. Внутренний цикл (по J) при каждом фиксированном во внешнем цикле номере строки I обеспечивает перебор всех N столбцов таблицы и печать произведения $I*J$. После завершения внутреннего цикла внешний повторяется при новом значении I, а внутренний восстанавливается в первоначальном виде – J снова "пробегает" все значения от I до N. Обратите внимание на оператор в строке 90: он обеспечивает перевод печати на новую строку. И еще одно замечание: чтобы получить вывод таблицы Пифагора в приведенном на рис. 5.I виде, в операторе *PRINT* мы использовали функцию *TAB*, которая будет описана далее.

Пример 5.2. Требуется составить БЕЙСИК-программу для упорядочения массива $A(N)$ по возрастанию его элементов.

Соответствующая программа показана на рис. 5.2. В ней реализован алгоритм, основанный на следующей идее: последовательно сравниваются каждые два соседних элемента и меняются местами, если они не упорядочены по возрастанию. Эта операция выполняется во внутреннем цикле (по параметру K). За одно выполнение этого цикла максимальный из рассмотренных элементов оказывается на последнем месте ("выпадает в осадок") и в дальнейших сравнениях не участвует (индекс K меняется от I до $N-I$, где I – параметр внешнего цикла). За одно выполнение внешнего цикла количество сравниваемых во внутреннем цикле элементов массива сокращается на единицу. В общем случае так происходит, пока во внешнем цикле не будет просмотрен весь массив. Однако, если массив частично упорядочен, возможно "досрочное" завершение сортировки: если после просмотра на некотором шаге оставшихся элементов массива не было ни одной их перестановки, специальный параметр-признак P сохранит свое нулевое значение, установленное в начале внешнего цикла, и оператор I70 (рис. 5.2) в этом случае обеспечит принудительный выход из цикла и передачу управления операторам выдачи результатов.

```
10  REM СОРТИРОВКА СРАВНЕНИЕМ СМЕЖНЫХ ПАР
20  DIM A(10)
30  DATA 0,8,9,6,1,3,2,5,7,10,12
40  INPUT N:PRINT "N="N
50  FOR I=1 TO N
60    READ A(I)
65    PRINT A(I);
70  NEXT I
75  PRINT
80  FOR I=1 TO N
90    P=0
100   FOR K=1 TO N-I
110     IF A(K)<=A(K+1) THEN 160
120       Z=A(K)
130       A(K)=A(K+1)
140       A(K+1)=Z
150       P=1
160   NEXT K
170   IF P=0 THEN 183
180 NEXT I
183 PRINT:PRINT
185 PRINT "ОТСОРТИРОВАННЫЙ МАССИВ:"
186 PRINT
190 FOR I=1 TO N
200   PRINT A(I);
210 NEXT I
220 END
ПРОТОКОЛ РАБОТЫ:
N= 10
 0  8  9  6  1  3  2  5  7  10
ОТСОРТИРОВАННЫЙ МАССИВ:
0  1  2  3  5  6  7  8  9  10
```

Рис.5.2

Пример 5.3. Рассмотрим более сложный пример. Из всех участков массива $A(N)$, сплошь состоящих из нулей, выбрать самый длинный и отметить индексы его начала и конца.

Блок-схема (рис. 5.3,а) дает наглядное представление об алгоритме. Чтобы облегчить изучение приведенной на рис. 5.3,б программы, поясним назначение используемых переменных:

$M1$ – длина нулевого участка, на выходе – длина максимального участка;

$N1$, $K1$ – соответственно индексы начала и конца нулевого участка, на выходе – максимального по длине;

H, K – текущие значения индексов соответственно начала и конца нулевого участка;

L – счетчик количества нулей;

P – признак, сохраняющий значение $\emptyset$, если очередной выбранный элемент массива отличен от $\emptyset$, и I – в противном случае.

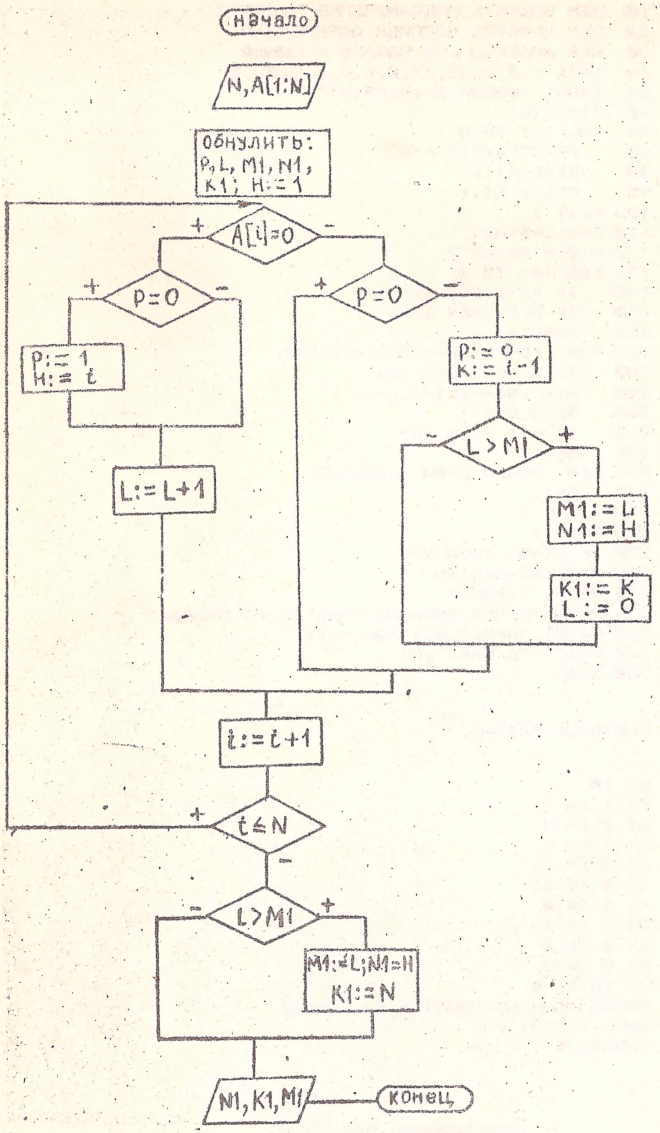
Если очередной элемент массива равен нулю, то при $P=0$ индекс начала нулевого участка H полагается равным I – текущему значению индекса массива, а P становится равным I (отмечаем, что "пошли" нули) и в счетчик L прибавляется единица; при $P \neq 0$ в этой ситуации в счетчик L сразу добавляется единица: это означает, что обрабатывается уже не первый элемент нулевого участка.

Как только $A(I) \neq 0$ и $P \neq 0$ (налицо переход от нулевого к ненулевому участку), определяется индекс конца нулевого участка ($K=I-1$) и признак P сбрасывается в нуль. Благодаря логическому блоку $L>M1$ в конечном счете определяется максимальный по длине нулевой участок.

Заслуживает внимания обработка ситуации, когда максимальный нулевой участок ("хвост") завершает массив (операторы 270–290).

Пример 5.4. Дана матрица $A(M,N)$ (т.е. содержащая M строк и N столбцов). Требуется найти минимальный элемент матрицы и записать нули в ту строку и столбец, где он находится.

Текст программы и протокол ее выполнения приведены на рис. 5.4. Исходная матрица задана четырьмя операторами $DATA$ (строки 40–70). Минимальный элемент находится последовательным сравнением каждого элемента матрицы с текущим минимумом $M1$, в качестве начального значения которого принят элемент $A(1,1)$. Если обнаруживается, что $A(I,J)<M1$ (т.е. найден элемент, числовое значение которого меньше текущего минимума), то текущее значение $M1$ заменяется на $A(I,J)$. Одновременно запоминаются номера строки ($K=I$) и столбца ($L=J$), соот-


начало
N,A[1:N]
обнулить:
P, L, M1, N1,
K1; H:=1
A[i]=0
+
-
P=0
P:=1
H:=i
L:=L+1
P=0
P:=0
K:=i-1
L>M1
M1:=L
N1:=H
K1:=K
L:=0
i:=i+1
i≤N
L>M1
M1:=L; N1:=H
K1:=N
N1, K1, M1
конец

а

Рис. 5.3

```
10  REM ВЫБРАТЬ МАКСИМАЛЬНЫЙ НУЛЕВОЙ
20  REM УЧАСТОК МАССИВА. ОПРЕДЕЛИТЬ
30  REM ИНДЕКСЫ ЕГО НАЧАЛА И КОНЦА
40  DATA 0,0,0,25,31,0,0,0,0,9,55,0
50  INPUT "ВВЕДИ N";N:PRINT"N="N
60  DIM A(N)
70  FOR I=1 TO N
80    PRINT"A("I")=";
90    READ A(I)
95    PRINT A(I)
100 NEXT I
110 P=0:L=0:H=1
120 M=0:N1=0:K1=0
130 FOR I=1 TO N
140   IF A(I)=0 THEN 210
150   IF P=0 THEN 260
160   P=0:K=I-1
170 REM  КОНЕЦ НУЛЕВОГО УЧАСТКА
180   IF L<=M1 THEN 260
190   M1=L:N1=H:K1=K:L=0
200   GOTO 260
210   IF P=1 THEN 250
220   H=I
230 REM  ПОШЛИ НУЛИ H-НАЧАЛО
240   P=1
250   L=L+1
260 NEXT I
270 IF L<=M1 THEN 290
280 M1=L:N1=H:K1=N
285 PRINT "ХВОСТ"
290 PRINT"САМЫЙ ДЛИННЫЙ УЧАСТОК ИЗ НУЛЕЙ:"
300 PRINT "НАЧ:";N1;"КОН:";K1
310 PRINT "ДЛИНА:";M1
320 END
```

ПРОТОКОЛ РАБОТЫ:

```
N= 10
A( 1 )= 0
A( 2 )= 0
A( 3 )= 0
A( 4 )= 25
A( 5 )= 31
A( 6 )= 0
A( 7 )= 0
A( 8 )= 0
A( 9 )= 0
A( 10 )= 9
САМЫЙ ДЛИННЫЙ УЧАСТОК ИЗ НУЛЕЙ:
НАЧ: 6 КОН: 9
ДЛИНА: 4
```

б

Продолжение рис. 5.3

ветствующих промежуточному минимальному элементу. Ясно, что после завершения этой процедуры в поле переменной *M1* будет находиться значение минимального элемента матрицы. В последующих операторах программы строка и столбец матрицы, на пересечении которых находится ее минимальный элемент, заполняются нулями, а минимальный элемент восстанавливается на своем месте.

```
10  REM НАЙТИ MIN ЭЛЕМЕНТ МАТРИЦЫ
20  REM И ЗАПИСАТЬ НУЛИ В ТУ СТРОКУ
30  REM И СТОЛБЕЦ,ГДЕ ОН НАХОДИТСЯ
40  DATA 1,1,1,1,1
50  DATA 2,2,2,2,2
60  DATA 3,3,-3,3,3
70  DATA 4,4,4,4,4
80  M=4:N=5
90  FOR I=1 TO M
100   FOR J=1 TON
110     READ A(I,J)
120   NEXT J
130 NEXT I
140 M1=A(1,1):K=1:L=1
150 FOR I=1 TO M
160   FOR J=1 TO N
170     IF A(I,J)>=M1 THEN 190
180     M1=A(I,J):K=I:L=J
190   NEXT J
200 NEXT I
210 FOR J=1 TO N:A(K,J)=0:NEXT J
220 FOR I=1 TO M:A(I,L)=0:NEXT I
230 A(K,L)=M1
240 FOR I=1 TO M
250   FOR J=1 TO N
260     PRINT A(I,J);
270   NEXT J
280 PRINT
290 NEXT I
300 END
```

ПРОТОКОЛ РАБОТЫ:

```
 1  1  0  1  1
 2  2  0  2  2
 0  0 -3  0  0
 4  4  0  4  4
```

Рис. 5.4

Пример 5.5. Снова рассмотрим матрицу $A(N,N)$. Составим программу ее транспортирования (т.е. обмена местами ее строк и столбцов).

Текст программы и протокол ее выполнения приведены на рис. 5.5. Прежде всего, замечаем, что при транспортировании элементы главной диагонали должны оставаться без движения и надо только обменять местами элементы матрицы, расположенные симметрично относительно главной диагонали (мы уже не говорим о том, что операция "обмена" вида $A(I,J) = A(J,I)$ в принципе неверна): $D=A(I,J):A(I,J)=A(J,I):A(J,I)=D$

```
10  REM ТРАНСПОНИРОВАНИЕ МАТРИЦ
20  DATA 11,12,13,14
30  DATA 21,22,23,24
40  DATA 31,32,33,34
50  DATA 41,42,43,44
60  N=4
70  FOR I=1 TO N
80    FOR J=1 TO N
90      READ A(I,J)
100     PRINT A(I,J);
110   NEXT J
120   PRINT
130 NEXT I
140 FOR  I=2 TO N
150   FOR J=1 TO I-1
160     D=A(I,J)
170     A(I,J)=A(J,I)
180     A(J,I)=D
190   NEXT J
200 NEXT I
210 PRINT:PRINT"ТРАНСПОНИРОВАННАЯ"
220 FOR I=1 TO N
230   FOR J=1 TO N
240     PRINT A(I,J);
250   NEXT J
260   PRINT
270 NEXT I
280 END


ПРОТОКОЛ РАБОТЫ:


 11  12  13  14
 21  22  23  24
 31  32  33  34
 41  42  43  44

ТРАНСПОНИРОВАННАЯ
 11  21  31  41
 12  22  32  42
 13  23  33  43
 14  24  34  44
```

Рис. 5.6

Однако при более внимательном рассмотрении этой процедуры становится очевидным, что ее нельзя выполнять для всех значений I и J, иначе каждый элемент матрицы будет участвовать в обмене дважды: один раз как $A(I,J)$ -элемент, и еще один раз - как $A(J,I)$ -элемент. В результате такого двойного обмена исходная матрица останется без изменения. Так что операцию обмена следует выполнять только для элементов, стоящих, например, ниже главной диагонали (для стоящих выше диагонали рассуждения аналогичны). Именно поэтому в теле программы внешний цикл начинается с $I = 2$, а параметр J изменяется во внутреннем цикле от 1 до $I - 1$. Сложные циклы наиболее часто используют при обработке двумерных массивов-матриц. Но не следует думать, что максимальная глубина вложенности циклов равна двум. В интерпретаторе описываемого БЕЙСИКа она равна семи.

Пример 5.6. Найти все "счастливые" автобусные билеты среди билетов с номерами от 000000 до *NNNNNN* и подсчитать их количество.

Программа для решения этой задачи и протокол ее выполнения при $N = 2$ приведены на рис. 5.6.

Упражнения

1. Что будет выдано на экран дисплея в результате выполнения следующей программы (для определенности принять $N = 5$):

```
10  INPUT N
20  DIM A(N,N)
30  FOR I=1 TO N
40  FOR J=1 TO N
50    IF I=J THEN A(I,J)=1:GOTO 70
60    A(I,J)=0
70    PRINT A(I,J);
80   NEXT J
90   PRINT
100 NEXT I
110 END
```

2. Для заданной матрицы $A(N,N)$ (N - константа) найти:

а) номер строки, содержащей наибольшее число ненулевых элементов;

б) номер столбца, содержащего наименьшее число нулевых элементов;

в) номер строки, среднеарифметическое положительных элементов которой является наименьшим;

г) номер столбца, в котором наибольшее количество элементов, значение которых совпадает с номером столбца;

д) $max\ min\ |a_{ij}|$;

е) "след" (сумму диагональных элементов) матрицы.

```
10  REM СЧАСТЛИВЫЕ БИЛЕТЫ СРЕДИ
20  REM БИЛЕТВ С НОМЕРАМИ Ø-NNNNNN
30  PRINT "ВВЕДИ N"
40  INPUT N: PRINT"N=";N
45  E5=100000!:E4=10000:E3=1000
46  E2=100:E1=10
50  KOL=Ø
60  FOR I=1 TO N
70  FOR J=1 TO N
80  FOR K=1 TO N
90  FOR L=1 TO N
100 FOR M=1 TO N
110 FOR N1=1 TO N
130 IF (I+J+K)<>(L+M+N1) THEN 170
140 KOL=KOL+1
150 S=I*E5+J*E4+K*E3+L*E2+M*E1+N1
160 PRINT"СЧАСТЛИВЫЙ:"S
170 NEXT N1
180 NEXT M
190 NEXT L
200 NEXT K
210 NEXT J
220 NEXT I
230 PRINT "     ВСЕГО :"KOL
240 END
```

ПРОТОКОЛ РАБОТЫ:

```
ВВЕДИ N
N= 2
СЧАСТЛИВЫЙ:  111111
СЧАСТЛИВЫЙ:  112112
СЧАСТЛИВЫЙ:  112121
СЧАСТЛИВЫЙ:  112211
СЧАСТЛИВЫЙ:  121112
СЧАСТЛИВЫЙ:  121121
СЧАСТЛИВЫЙ:  121211
СЧАСТЛИВЫЙ:  122122
СЧАСТЛИВЫЙ:  122212
СЧАСТЛИВЫЙ:  122221
СЧАСТЛИВЫЙ:  211112
СЧАСТЛИВЫЙ:  211121
СЧАСТЛИВЫЙ:  211211
СЧАСТЛИВЫЙ:  212122
СЧАСТЛИВЫЙ:  212212
СЧАСТЛИВЫЙ:  212221
СЧАСТЛИВЫЙ:  221122
СЧАСТЛИВЫЙ:  221212
СЧАСТЛИВЫЙ:  221221
СЧАСТЛИВЫЙ:  222222
    ВСЕГО :  20
```

Рис. 5.6

3. Построить матрицу вида

$$\begin{pmatrix} n & n-1 & n-2 & \dots & 1 \\ n+1 & n & n-1 & \dots & 2 \\ n+2 & n+1 & n & \dots & 3 \\ \hline 2n-1 & 2n-2 & 2n-3 & \dots & n \end{pmatrix}$$

4. Дан вектор $X(10)$. Построить матрицу $Y(10,10)$ следующего вида:

а)
$$\begin{pmatrix} X_1 & X_2 & X_3 & \dots & X_{10} \\ X_2 & X_1 & X_2 & \dots & X_9 \\ X_3 & X_2 & X_1 & \dots & X_8 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ X_9 & X_8 & X_7 & \dots & X_2 \\ X_{10} & X_9 & X_8 & \dots & X_1 \end{pmatrix}$$

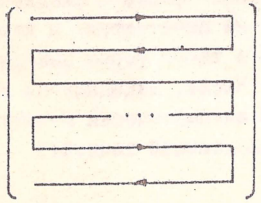
б)
$$\begin{pmatrix} X_1 & X_2 & \dots & X_9 & X_{10} \\ X_2 & X_3 & \dots & X_{10} & X_1 \\ X_3 & X_4 & \dots & X_1 & X_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ X_{10} & X_1 & & X_8 & X_9 \end{pmatrix}$$

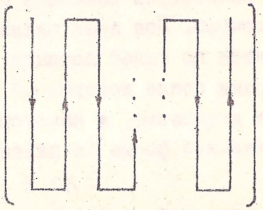
5. Построить матрицу $Y(10,10)$ вида:

а)
$$\begin{pmatrix} 1 & 20 & 21 & \dots & 81 & 100 \\ 2 & 19 & 22 & \dots & 82 & 99 \\ 3 & 18 & 23 & \dots & 83 & 98 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 9 & 12 & 29 & & 89 & 92 \\ 10 & 11 & 30 & & 90 & 91 \end{pmatrix}$$

б)
$$\begin{pmatrix} 1 & 3 & 5 & \dots & 19 \\ 2 & 4 & 6 & \dots & 20 \\ 21 & 23 & 25 & \dots & 39 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 82 & 84 & 86 & \dots & 100 \end{pmatrix}$$

6. Дана матрица $A(N,N)$. Напечатать ее элементы, начиная с $A(1,1)$, в таком порядке:

а)


б)


7. В матрице $X(M,N)$ найти четыре минимальных элемента.

8. Среди элементов матрицы $A(M,N)$ найти наименьший положительный и наибольший отрицательный элементы.

9. Дано число X. Построить матрицу A (7,7) вида

а)
$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ X & X & X & \dots & X \\ X^2 & X^2 & X^2 & \dots & X^2 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ X^6 & X^6 & X^6 & \dots & X^6 \end{pmatrix}$$

б)
$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & X & X & \dots & X \\ 1 & X & X^2 & \dots & X^2 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & X & X^2 & \dots & X^6 \end{pmatrix}$$

10. В матрице A (10,5) поменять местами соседние строки матрицы (первую со второй, третью – с четвертой и т.д.).

6. ВЫВОД ГРАФИКОВ, ТАБЛИЦ. УПРАВЛЕНИЕ ВЫВОДОМ

В разд 2.2 упоминалось, что разделители в списке вывода оператора *PRINT* – запятая и точка с запятой – служат и для элементарного управления печатью. А именно, запятая как разделитель обеспечивает вывод в зонном, а точка с запятой – в плотном формате. Кроме того, применение точки с запятой в конце выводного списка всегда обусловливает (при повторении того же оператора *PRINT* или при переходе к новому оператору печати) продолжение вывода на прежней строке. Перехода на новую строку можно достичь употреблением в подходящем месте "пустого" оператора *PRINT* (без списка вывода). Рассмотрим более гибкие средства вывода числовой и текстовой информации.

6.1. Вывод числовой информации

Формат печати числа зависит от его величины. Числа из диапазона (0.1,10) печатаются обычно в естественной форме с фиксированной десятичной точкой. Под знак числа (плюс не печатается) и десятичную точку отводится по одной позиции. Числа в такой форме выводятся с 12–14 цифрами после десятичной точки. Числа, находящиеся вне указанного выше диапазона, в некоторых ЭВМ автоматически печатаются в экспоненциальной форме (с плавающей десятичной точкой):

$$\pm m_x E \pm pp,$$

где $\pm m_x$ – мантисса числа X, $1 \leqslant m_x < 10$.

Формирование таблиц с помощью символа "точка с запятой" не дает удобного воспроизведения результата: столбцы оказываются деформированными, поскольку вывод чисел нередко осуществляется с различным количеством цифр (рис. 6.I).

```
10  PRINT "ТАБЛИЦА УМНОЖЕНИЯ"
15  PRINT:PRINT
20  FOR I=2 TO 6 STEP 4
30   FOR J=1 TO 10
40     I1=I+1:I2=I+2:I3=I+3
50     PRINT I"X"J"="I*J;
60     PRINT I1"X"J"="I1*J;
70     PRINT I2"X"J"="I2*J;
80     PRINT I3"X"J"="I3*J;
85     PRINT
90    NEXT J
100 PRINT:PRINT
110 NEXT I
120 END
```

ПРОТОКОЛ РАБОТЫ:

ТАБЛИЦА УМНОЖЕНИЯ

```
2 X 1 = 2   3 X 1 = 3   4 X 1 = 4   5 X 1 = 5
2 X 2 = 4   3 X 2 = 6   4 X 2 = 8   5 X 2 = 10
2 X 3 = 6   3 X 3 = 9   4 X 3 = 12  5 X 3 = 15
2 X 4 = 8   3 X 4 = 12  4 X 4 = 16  5 X 4 = 20
2 X 5 = 10  3 X 5 = 15  4 X 5 = 20  5 X 5 = 25
2 X 6 = 12  3 X 6 = 18  4 X 6 = 24  5 X 6 = 30
2 X 7 = 14  3 X 7 = 21  4 X 7 = 28  5 X 7 = 35
2 X 8 = 16  3 X 8 = 24  4 X 8 = 32  5 X 8 = 40
2 X 9 = 18  3 X 9 = 27  4 X 9 = 36  5 X 9 = 45
2 X 10 = 20  3 X 10 = 30  4 X 10 = 40  5 X 10 = 50


6 X 1 = 6   7 X 1 = 7   8 X 1 = 8   9 X 1 = 9
6 X 2 = 12  7 X 2 = 14  8 X 2 = 16  9 X 2 = 18
6 X 3 = 18  7 X 3 = 21  8 X 3 = 24  9 X 3 = 27
6 X 4 = 24  7 X 4 = 28  8 X 4 = 32  9 X 4 = 36
6 X 5 = 30  7 X 5 = 35  8 X 5 = 40  9 X 5 = 45
6 X 6 = 36  7 X 6 = 42  8 X 6 = 48  9 X 6 = 54
6 X 7 = 42  7 X 7 = 49  8 X 7 = 56  9 X 7 = 63
6 X 8 = 48  7 X 8 = 56  8 X 8 = 64  9 X 8 = 72
6 X 9 = 54  7 X 9 = 63  8 X 9 = 72  9 X 9 = 81
6 X 10 = 60  7 X 10 = 70  8 X 10 = 80  9 X 10 = 90
```

Рис. 6.I

6.2. Табуляторная функция. Графики

Стройную печать таблиц, вывод графиков можно осуществить с помощью табуляторной функции *TAB*. Она обеспечивает наиболее гибкие возможности по управлению выводом. Употребляется функция *TAB* только в операторе *PRINT*. Формат функции:

TAB (выражение)

Целая часть задаваемого в скобках произвольного арифметического выражения (им, в частности, может быть имя переменной или константа) определяет номер позиции в печатаемой строке, начиная с которой выдается в печатаемую строку следующий за *TAB* элемент списка вывода. Тем самым фиксируется и положение остальных элементов из выводимой последовательности. Номер необходимой позиции всегда отсчитывается от начала печатной строки. Список элементов вывода одного оператора *PRINT* может содержать несколько указателей табуляторной функции, отделяемых или неотделяемых друг от друга и от других элементов вывода разделителями (запятая, точка с запятой). Надо только иметь в виду, что задаваемые ими номера позиций вывода должны образовывать неубывающую последовательность.

Рассмотрим соответствующие примеры.

Пример 6.1. Вывести на печать равнобедренный треугольник заданной высоты.

Проанализируйте приведенную на рис. 6.2 программу и результаты ее работы при $H = 14$.

Пример 6.2. Составить программу формирования и выдачи на печать треугольника Паскаля.

Соответствующая программа и результат ее работы приведены на рис. 6.3. В [2] приводятся алгоритмы решения этой задачи, основанные обычно на формировании двумерного массива, в котором каждая последующая строка вычисляется на основе предыдущей по такому простому правилу: каждый ее элемент (не считая крайних, всегда равных единице) равен сумме двух над ним стоящих из предыдущей строки.

Мы здесь даем алгоритм, основанный на том, что каждая строка треугольника - биномиальные коэффициенты разложения $(a+b)^n$, $n = 0, 1...$ В программе переменная A определяет позицию вершины треугольника, T - ширину поля вывода для каждого числа, L - шаг "лестницы" (рекомендуется $L \simeq T/2$). Перед запуском программы на микроЭВМ "Электроника ДЗ-28" следует выдать директиву *PRINT!5·Ø!*, которая в дальнейшем обеспечит отсечение нулевой дробной части при выводе целых чисел.

```
10 REM ВЫВОД ТРЕУГОЛЬНИКА
20 INPUT H
30 PRINT "ВЫСОТА H=";H
35 PRINT TAB(20)"*"
40 FOR K=1 TO H-1
50   PRINT TAB(20-K)"*"TAB(20+K)"*"
60 NEXT K
70 FOR X=20-H TO 20+H
80   PRINT TAB(X)"*";
90 NEXT X
99 END
```

ПРОТОКОЛ РАБОТЫ:

```
ВЫСОТА H= 14
                    *
                   * *
                  *   *
                 *     *
                *       *
               *         *
              *           *
             *             *
            *               *
           *                 *
          *                   *
         *                     *
        *                       *
       *************************
```

Рис. 6.2

```
20 PRINT "ВВЕДИ N"
30  INPUT N:PRINT"N=";N
40  A=18:L=3:T=6
50 PRINT TAB(A-7)"ТРЕУГОЛЬНИК ПАСКАЛЯ"
60 PRINT
70 PRINT TAB(A+1)"1"
80  FOR I=1 TO N
90   C=1
100 PRINT TAB(A-L*I)"1";
110  FOR K=1 TO I
120   C=C*(I-K+1)/K
130  PRINT TAB(A-L*I+T*K)C;
140  NEXT K:PRINT
150 NEXT I
160 END
```

ПРОТОКОЛ РАБОТЫ:

```
ВВЕДИ N
N= 6
              ТРЕУГОЛЬНИК ПАСКАЛЯ
                        1
                     1     1
                  1     2     1
               1     3     3     1
            1     4     6     4     1
         1     5    10    10     5     1
      1     6    15    20    15     6     1
```

Рис. 6.3

На рис. 6.4 – 6.6 изображены программы вывода графиков функций $y=AX^2$, $y=AX^3$, $y=e^{-x}\cdot\sin 2\pi x$. Используя приведенные сразу после текста каждой программы результаты ее работы, разберите самостоятельные их алгоритмы. Учтите только, что ось X направлена вертикально, а ось y – горизонтально. Точки графика выводятся на печать по мере возрастания аргумента X, являющегося параметром цикла, и располагаются относительно оси X в соответствии со значением аргумента табуляторной функции $TAB(Y)$.

6.3. Укрупненное начертание символов

Укрупненное начертание какого-либо символа (например, буквы русского алфавита) связано с заданием этого символа табличкой, заполненные клетки которой образуют контур (рис. 6.7). Содержимое такой таблички можно закодировать с помощью I (клетка заполнена) и Ø (клетка не заполнена).

```
10 REM ГРАФИК Y=A*X^3
20  INPUT B,H,A:PRINT"B="B"H="H"A="A
30  PRINT:PRINT TAB(18)"!X"
40  FOR X=-B TO B STEP H
50    Y=A*X^3+18
60    IF X>0 THEN 130
70    IF X<0 THEN 150
80    FOR I=1 TO 36
90      PRINT TAB(I)"-";
100   NEXT I
110   PRINT TAB(37)"Y"
120   GOTO 160
130   PRINT TAB(18)"!"TAB(Y)"*"
140   GOTO 160
150   PRINT TAB(Y)"*"TAB(18)"!"
160 NEXT X
170 END
```

ПРОТОКОЛ РАБОТЫ:

```
B= 3 H= .5 A= .6
                    !X
 *                  !
         *          !
              *     !
                 *  !
                    *!
                    *!
-----------------------------------Y
                    !*
                    !*
                    ! *
                    !   *
                    !      *
                    !          *
```

Рис. 6.4

```
10  REM  ГРАФИК У=АЖХ^2
20  INPUT B,A,H:PRINT"B="B;"A="A;"H="H
30  PRINT"!X"
40  FOR X=-B TO B STEP H
50  Y=AЖX^2
60  IF X<>0 THEN 110
70  FOR I=1 TO 35
80    PRINT TAB(I)"-";
90  NEXT I
100 PRINT TAB(36)"Y":GOTO 120
110 PRINT"!"TAB(Y)"Ж"
120 NEXT X
130 END
```

ПРОТОКОЛ РАБОТЫ:

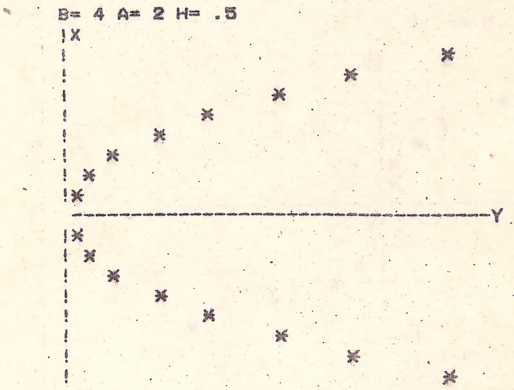


Рис. 6.5

```
5 'ГРАФИК ФУНКЦИИ Y=EXP(-X)*SIN(6.28*X)
10 M=28
20 D=.08
30 P2=6.28318
40 Z=20
50 FOR I=1 TO M
60   X=I*D
70 Y=EXP(-X)*SIN(P2*X)
80 T=INT(18*Y)+Z
90 PRINT TAB(T)"*"
100 NEXT I
110 END
```

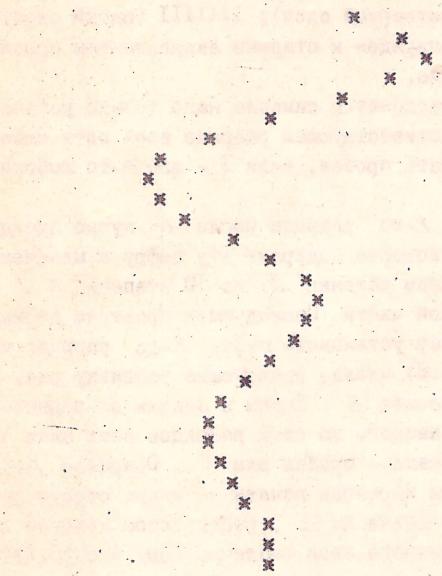


Рис. 6.6

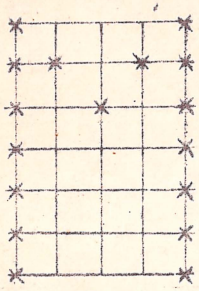


Рис. 6.7

Если исходная табличка состоит из 7 X 5 элементов (рис. 6.7), то для полной идентификации символа будет достаточно пяти семиразрядных десятичных чисел. Каждое из них будет кодировать один из пяти вертикальных слоев исходной таблички. Например, начертание символа *M* (рис. 6.7) кодируется следующими пятью числами: IIIIIII (первый вертикальный слой исходной таблички); I0 (второй слой); I00 (третий слой); I0 (четвертый слой); IIIIIII (пятый слой), причем, переход от младших разрядов к старшим эквивалентен просмотру исходной таблички сверху вниз.

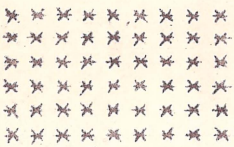
Для последовательной распечатки символа надо только установить, какие цифры содержатся в соответствующем разряде всех пяти чисел. Если это 0, следует отпечатать пробел, если I – какой-то выбранный символ (например *).

Для определения цифры *K*-го разряда числа *B* нужно прежде всего перейти к числу *M*, которое содержит эту цифру в младшем разряде. Это достигается путем деления *B* на I0 степени *K-1* и выделении из результата целой части. Последующая проверка делимости *M* нацело на I0 и позволяет установить цифру *K*-го разряда числа *B*. В программе (рис. 6.8) числа, кодирующие табличку рис. 6.7, присваиваются элементам массива *B*. Затем в циклах по переменным организуется проверка цифр каждого из семи разрядов всех пяти чисел и печать соответствующего знака – пробел или *. Оператор *PRINT* (после *NEXT L*) введен для перевода печати на новую строку по завершении работы внутреннего цикла по *L*, результатом которой является вывод одного горизонтального слоя таблички (см. рис. 6.7).

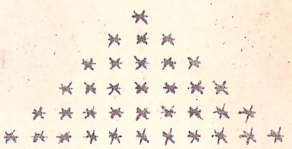
Упражнения

I. Составить программы выдачи на экран дисплея (на печать) следующих фигур:

а)



б)



в)

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
```

г)

```
* * * * * * * * * *
*                 *
*                 *
*                 *
*                 *
*                 *
* * * * * * * * * *
```

Размеры каждой из фигур заранее не фиксированы и задаются в операторе *INPUT* во время исполнения программы.

2. Составить программу для укрупненного начертания начальной буквы вашего имени.

3. Составить программу выдачи на печать (экран дисплея) графиков функций:

а) $y = \cos x$; б) $y = e^{-x} \cdot \ln \pi x$.

```
10 DIM B(5)
20 DATA 1111111,10,100,10,1111111
30 FOR L=1 TO 5
40   READ B(L)
50  NEXT L
60  FOR K=1 TO 7
70  FOR L=1 TO 5
80  M=INT(B(L)/10^(K-1))
90  M=M-INT(M/10)*10
100 IF M=0 THEN PRINT " ";
110 IF M<>0 THEN PRINT "*";
120 NEXT L
130 PRINT
140 NEXT K
150 END
```

ПРОТОКОЛ РАБОТЫ:

```
*   *
**  **
* * *
*   *
*   *
*   *
*   *
```

Рис. 6.8

7. НЕСТАНДАРТНЫЕ ФУНКЦИИ И ПОДПРОГРАММЫ

7.1. Функции пользователя

Удобства, связанные с использованием стандартных функций БЕЙСИКа, известны. Язык позволяет пользователю создавать свои, нестандартные функции. Запись

$$DEF\ FN\alpha(X)=f(X,A_1,A_2,\ldots,A_n)$$

определяет нестандартную функцию. Здесь DEF – ключевое слово, которое служит для описания нестандартной функции; $FN\alpha$ – имя функции, в котором две первые буквы фиксированы; α – одна из букв латинского алфавита, выбираемая произвольно; X – аргумент (формальный параметр) определяемой функции (он может быть представлен любой простой переменной); f – любое арифметическое выражение, задающее вид функции (оно может содержать формальный параметр X и (или) глобальные переменные A_1, A_2, ..., A_n). В программе могут быть описаны несколько нестандартных функций. Все описания помещают обычно в начале программы. При последовательном выполнении операторов описания функций игнорируются. Вычисление значения функции происходит в те моменты, когда в программе встречается указатель функции $FN\alpha(A)$, т.е. ее имя с заключенным в скобки фактическим параметром. Переменные A_1, A_2, ..., A_n, если они есть в определении функции, должны быть инициализированы до первого использования указателя функции. Указатель функции $FN\alpha(A)$ может встречаться как операнд выражения, как элемент списка вывода оператора $PRINT$, т.е. ведет себя как простая или индексированная переменная. В качестве фактического параметра допускаются выражения.

Пример 7.1. В качестве примера, иллюстрирующего применение нестандартной функции, рассмотрим задачу вычисления интеграла по квадратурной формуле Симпсона:

$$J=\int_a^b f(x)\,dx\approx\frac{b-a}{3n}(f_0+4)(f_1+f_3+\ldots+f_{n-1})+2(f_2+\ldots+f_{n-2})+f_n),$$

представляющей интеграл через сумму значений подынтегральной функции в некоторых точках X_i – узлах квадратурной формулы. Здесь

$$f_i=f(a+ih),\ i=0,1,\ldots,n;\ h=(b-a)/n,\ n=2^k.$$

Число узловых точек равно $n+1$; h – расстояние между двумя соседними узловыми точками. Значение интеграла аппроксимируется последовательностью S_1 , S_2 , S_3 ,..., S_K , которая сходится, если функция f достаточно гладкая и если арифметические операции выполняются точно.

На каждом шаге вычислений число узловых точек удваивается. Чтобы избежать вычисления функции $f(x)$ 2^K раз на каждом K-м шаге (поскольку в принципе можно использовать значения f_i , полученные на предыдущих шагах), воспользуемся приемом, предложенным в [4]. В сумме S_K три члена:

$$S_K = S_K^{(1)} + S_K^{(2)} + S_K^{(4)},$$

которые обозначают суммы в узловых точках с весами 1, 2 и 4. Их можно определить с помощью следующих рекуррентных соотношений (для $K > 1$):

$$S_K^{(1)} = \frac{1}{2} S_{K-1}^{(1)} ;$$

$$S_K^{(2)} = \frac{1}{2} S_{K-1}^{(2)} + \frac{1}{4} S_{K-1}^{(4)} ;$$

$$S_K^{(4)} = \frac{4h}{3}\left(f(a+h)+f(a+3h)+\ldots+f(a+(n-1)h)\right),$$

причем, начальные значения равны соответственно:

$$S_1^{(1)} = \frac{h}{3}\left(f(a)+f(b)\right), \quad S_1^{(2)} = 0,$$

$$S_1^{(4)} = \frac{4h}{3} f\left(\frac{a+b}{2}\right).$$

Составим программу вычисления определенного интеграла методом Симпсона, в которой подынтегральная функция оформлена как нестандартная функция БЕЙСИКа $FNF(X)$. Для тестового примера возьмем интеграл

$$\int_0^1 \frac{dx}{1+x^2},$$

который, как известно, равен $arctg\,1 = \pi/4$. Соответствующая программа и результаты ее работы приведены на рис. 7.1. Нестандартная функция, представляющая подынтегральное выражение, в программе определяется в строке 2Ø, используется (вызывается) в строках 5Ø, 7Ø, 13Ø (в цикле), всякий раз с новыми значениями аргумента.

```
10  REM ВЫЧИСЛЕНИЕ ИНТЕГРАЛА ПО СИМПСОНУ
20  DEF FNF(X)=1/(1+X*X)
25  INPUT A,B,E
30  PRINT"ПРЕДЕЛЫ ИНТЕГРИРОВАНИЯ:";A;B
35  PRINT "ТОЧНОСТЬ:";E
40  N=2:H=(B-A)*.5
50  S1=H*(FNF(A)+FNF(B))
60  S2=0
70  S4=4*H*FNF(A+H)
80  S=S1+S2+S4
90  SS=S:N=2*N:H=H/2
100 S1=.5*S1
110 S2=.5*S2+.25*S4
120 S4=0:I=1
130 S4=S4+FNF(A+I*H):I=I+2
140 IF I<=N THEN 130
150  S4=S4*4*H
160  S=S1+S2+S4
170 IF ABS(S-SS)>E THEN 90
180 S=S/3
190 PRINT "ИНТЕГРАЛ=",S
200 END


ПРОТОКОЛ РАБОТЫ:


ПРЕДЕЛЫ ИНТЕГРИРОВАНИЯ: 0  1
ТОЧНОСТЬ: 1E-05
ИНТЕГРАЛ=        .785398162862
```

Рис. 7.1

Чтобы вычислить другой интеграл, например

$$\int_0^{\frac{\pi}{2}} \frac{dx}{\sqrt{a^2\cos^2 x + b^2 \sin^2 x}},$$

достаточно заменить 20 строку программы на следующую:

```
20 DEF FNF(X)=1/SQRT((a*COS(X))^2+(B*Sin(X))^2).
```

Пределы интегрирования и точность вычисления интеграла вводятся в момент выполнения программы в оператор 25.

7.2. Подпрограммы

Подпрограмма – это специальным образом оформленная последовательность операторов языка, которая выполняет четко определенную функцию и хранится только в одном месте программы (либо в специальной

библиотеке подпрограмм), однако ее можно вызывать для выполнения одной или нескольких точек программы. За счет хранения подпрограмм отдельно от основной программы объем последней уменьшается. Вместо того, чтобы повторять всю последовательность операторов каждый раз, когда в этом возникает необходимость, в программу включают один оператор для вызова требуемой подпрограммы. Но главное преимущество подпрограмм проявляется в современной технологии программирования больших и сложных научно-технических проблем. По мере прогресса в искусстве программирования программы стали создаваться методом пошаговой детализации (последовательного уточнения), согласно которому программируемая задача разбивается на некоторое число подзадач, которые в последующем программируются в виде программы с отдельными подпрограммами. Такая организация облегчает проверку правильности программ, улучшает их читаемость.

Начало подпрограммы в БЕЙСИКе никак специально не оформляется: формально подпрограммой является любая последовательность операторов языка, которая завершается оператором *RETURN*. Он служит признаком конца подпрограммы.

Обращение к подпрограмме осуществляется с помощью оператора *GOSUB*, имеющего такой формат:

GOSUB <номер строки>

Встретив в программе этот оператор, ЭВМ выполняет следующие действия:

1) управление передается оператору той строки программы, номер которой указан после служебного слова *GOSUB* (первому оператору подпрограммы);

2) выполняются операторы подпрограммы;

3) встретив оператор *RETURN*, машина возвращает управление оператору, стоящему за последним выполненным *GOSUB*.

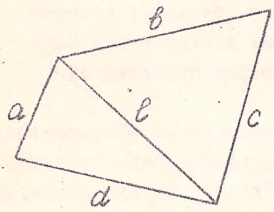
Оператор *GOSUB* – основной инструмент структурного программирования в БЕЙСИКе. Методом нисходящего проектирования программист расчленяет программу на подпрограммы, используя *GOSUB* для вызова каждого сложного фрагмента (подпрограммы). В качестве подпрограммы в стадии отладки можно вводить "заглушки" типа

```
1000 PRINT "ЗДЕСЬ БУДЕТ ПОДПРОГРАММА"
1100 RETURN
```

Затем, будучи уверенным в правильности логического построения программы в целом, программист детально разрабатывает и "расписывает" каждую подпрограмму, заменяя ею соответствующую "заглушку".

Пример 7.2. Вычислить площадь четырехугольника, если известны длины его сторон a, b, c, d и одна из диагоналей ℓ.

Диагональ делит четырехугольник на два треугольника с известными сторонами. Площадь каждого из них ($S1$ и $S2$ соответственно) найдем по формуле Герона. Для вычисления площади составим подпрограмму. В ней используются в качестве сторон треугольника формальные параметры x, y, z, которым при каждом вызове подпрограммы присваиваются конкретные значения. В основной программе предусмотрен ввод исходных данных, последовательное двукратное обращение к подпрограмме и вывод результатов (рис. 7.2).




```
10  REM ИСПОЛЬЗОВАНИЕ ПОДПРОГРАММ
20  INPUT A,B,C,D,L
30  PRINT"СТОРОНЫ: "A;B;C;D
40  PRINT"ДИАГОНАЛЬ:"L
50  X=A:Y=D:Z=L:GOSUB 120
60  S1=S
80  X=B:Y=C:GOSUB 120
90  S=S+S1
100 PRINT"ПЛОЩАДЬ ЧЕТЫРЕХУГОЛЬНИКА="S
110 STOP
120 REM
130 REM ПОДПРОГРАММА
140 REM РЕАЛИЗУЕТ ФОРМУЛУ ГЕРОНА
150 P=(X+Y+Z)/2
160 S=SQR(P*(P-X)*(P-Y)*(P-Z))
170 RETURN
```

```
ПРОТОКОЛ РАБОТЫ:

СТОРОНЫ:  3  4  3  4
ДИАГОНАЛЬ: 5
ПЛОЩАДЬ ЧЕТЫРЕХУГОЛЬНИКА= 12
```

Рис. 7.2

В подпрограммах можно выделить четыре типа используемых величин:

а) входные параметры подпрограммы – переменные, которые принимают различные значения при вызове подпрограммы из различных мест основной программы;

б) выходные параметры – переменные, значения которых определяются в ходе выполнения подпрограммы и являются результатом ее работы;

в) внешние переменные – переменные, которые определяются в основной программе и остаются неизменными при любом вызове подпрограммы;

г) внутренние переменные – переменные, используемые в подпрограмме как вспомогательные (параметры циклов, счетчики).

В качестве входных и выходных параметров подпрограммы удобно использовать такие переменные, которые для других целей в основной программе не используются. Перед обращением к подпрограмме входным (формальным) параметрам подпрограммы присваивают некоторые начальные значения, затем вызывают подпрограмму, а после ее завершения запоминают (используют в алгоритме) ее выходные параметры.

В программе на рис. 7.2 входные параметры подпрограммы – X, Y, Z; выходной параметр – S; ее внутренняя переменная – P.

Пример 7.3. Рассмотрим еще один способ построения треугольника Паскаля (см. пример 6.2). Напомним, что биномиальные коэффициенты разложения $(a+b)^i = C_i^0 a^i b^0 + C_i^1 a^{i-1} b + C_i^2 a^{i-2} b^2 + \ldots + C_i^i a^0 b^i$,

из которых формируется при различных i соответствующая i-я строка треугольника Паскаля, вычисляются по формуле

$$C_i^j = \frac{i!}{j!\,(i-j)!}, \quad i = 0,1,2,\ldots;\ j \le i.$$

Для вычисления $K!$ предусмотрим подпрограмму

Приведенная на рис. 7.3 программа демонстрирует многократный вызов подпрограммы в теле цикла. Подпрограмма имеет один входной параметр – K, которому последовательно присваиваются значения величин i, j, $(i-j)$, и для них подпрограмма вычисляет $K!$ Выходной параметр – F, который по завершении работы подпрограммы возвращает в основную программу соответствующий факториал. В основной программе вычисленные $i!$, $j!$ соответственно присваиваются вспомогательным переменным $C1$, $C2$.

```
10  REM БИНОМ
20  INPUT N:PRINT "N="N:PRINT
30  FOR I=0 TO N
40    K=I:GOSUB 190
50    C1=F
60    FOR J=0 TO I
70      K=J:GOSUB 190
80      C2=F
90      K=I-J:GOSUB 190
100     C=C1/C2/F
110     PRINT TAB(4*J)C;
120   NEXT J
130   PRINT
140 NEXT I
150 END
160 REM
170 REM ПОДПРОГРАММА
190 F=1:L=2
200 IF L>K THEN 230
210 F=F*L
220 L=L+1:GOTO 200
230 RETURN
```

ПРОТОКОЛ РАБОТЫ:

```
N= 9

 1
 1   1
 1   2   1
 1   3   3   1
 1   4   6   4   1
 1   5   10  10  5   1
 1   6   15  20  15  6   1
 1   7   21  35  35  21  7   1
 1   8   28  56  70  56  28  8   1
 1   9   36  84  126 126 84  36  9  1
```

Рис. 7.3

Упражнения

Составить программы, использующие нестандартные функции и подпрограммы, для решения следующих задач.

1. Вычислить и вывести на печать таблицу значений функции $f(x) = x^3 - 7x\cos x + 12.5$ и ее производной в интервале $(-\pi, \pi)$ с шагом 0,1.

2. Вычислить и напечатать максимальное значение функции $f(x) = (x-2)^2 e^{-(x-2)^2}$ в интервале (0,4) с точностью 0.01.

3. Вычислить интеграл от функции $f(x)=e^{-a\pi x}\ln(x+1)$ на отрезке [0, 2] по формуле трапеций:

$$\int_a^b f(x)dx=\frac{h}{2}\sum_{i=0}^{n-1}[f(a+(i+1)h)+f(a+ih)],\ \text{где}\ h=\frac{b-a}{n},$$

4. Вычислить число максимумов функции $f(x)=\sin 10x$ на интервале (0,5). Значения функции вычислять с точностью $\varepsilon = 0{,}01$.

ЛИТЕРАТУРА

1. Голубенцов А.Ф., Аникин В.М. Диалоговые алгоритмические языки ВЭЙСИК и ФОКАЛ: В 2 ч./ Под ред. А.М.Вогомолова. – Саратов: Изд-во Сарат. ун-та, 1983.

2. Дрейфус М., Гангloф К. Практика программирования на ФОРТРАНе. Упражнения с комментариями. – М.: Мир, 1978. – 224 с.

3. Ламуажье Ж.-П. Упражнения по программированию на ФОРТРАНе-IУ. – М.: Мир, 1978. – 162 с.

4. Вирт Н. Систематическое программирование. Введение. – М.: Мир, 1978. – 200 с.

5. Основы информатики и вычислительной техники: Пробное учеб. пособие для средних учебных заведений: В 2 ч. /Под ред. А.П.Ершова и В.М.Монахова, – К.: Рад. шк., 1985–1986.

6. Введение в программирование на алгоритмическом языке БЕЙСИК: Методические рекомендации к изучению школьного курса "Основы информатики и вычислительной техники" /Сост. М.П.Лапчик. – Омск: ОГПИ, 1985. – 68 с.

7. Агафонов В.Н., Поттосин И.В. и др. Сборник упражнений по программированию на языке ПАСКАЛЬ. – Новосибирск, 1985. – 80 с.

8. Макогон В.С. Основы алгоритмизации в информатике. – Одесса: ОГУ, 1986. – 72 с.

ОГЛАВЛЕНИЕ

Введение 3
I. Организация БЕЙСИК-программ. Основные конструкции языка 4
I.I. Подготовка БЕЙСИК-программы 5
I.2. Пример программы 7
I.3. Алфавит языка 8
I.4. Числа 9
I.5. Переменные и массивы I0
I.6. Стандартные функции языка II
I.7. Арифметические выражения I2
2. Основные операторы языка БЕЙСИК I5
2.I. Оператор присваивания I5
2.2. Простейший ввод и вывод информации I6
2.3. Операторы управления программой 2I
2.4. Дополнительные операторы ввода данных 34
3. Программирование циклических алгоритмов 43
3.I. Оператор цикла 56
4. Обработка таблиц (массивов) в БЕЙСИК-программе 59
4.I. Оператор описания массивов 59
4.2. Ввод и вывод массивов в БЕЙСИК-программе 60
5. Сложные циклы 65
6. Вывод графиков, таблиц. Управление выводом 76
6.I. Вывод числовой информации 76
6.2. Табуляторная функция. Графики 78
6.3. Укрупненное начертание символов 80
7. Нестандартные функции и подпрограммы 86
7.I. Функции пользователя 86
7.2. Подпрограммы 88
Литература 93

Владимир Сакович Макогон

Практика программирования на БЕЙСИКе

Учебное пособие

Редактор Л.В.Белоусова
Корректоры Н.И.Ефремова
И.П.Межировокий
Н.Н.Савченко
Н.О.Слонина

Св. доп. план, 1987, поз. 13

Под. к печ. 03.07.87 . БФ 07186 . Формат 60×84 1/16. Бумага тип. № 3 . Печать офсетная. Усл. печ. л. 5.58 . Усл. кр.-отт. 5.69 . Уч.-изд. л. 4.47 . Изд. № 5497 . Тираж 6000 . Зак. № 10044 . Цена 15 коп.

Одесский ордена Трудового Красного Знамени государственный университет им. И. И. Мечникова
270000, г. Одесса, ул. Петра Великого, 2

Межвузовское полиграфическое предприятие
252151, г. Киев, ул. Волынская, 60.

15 к.

Зак. 10044.